NICTA

Proof Engineering:
The Soft Side of Hard Proof

HCSS 2015

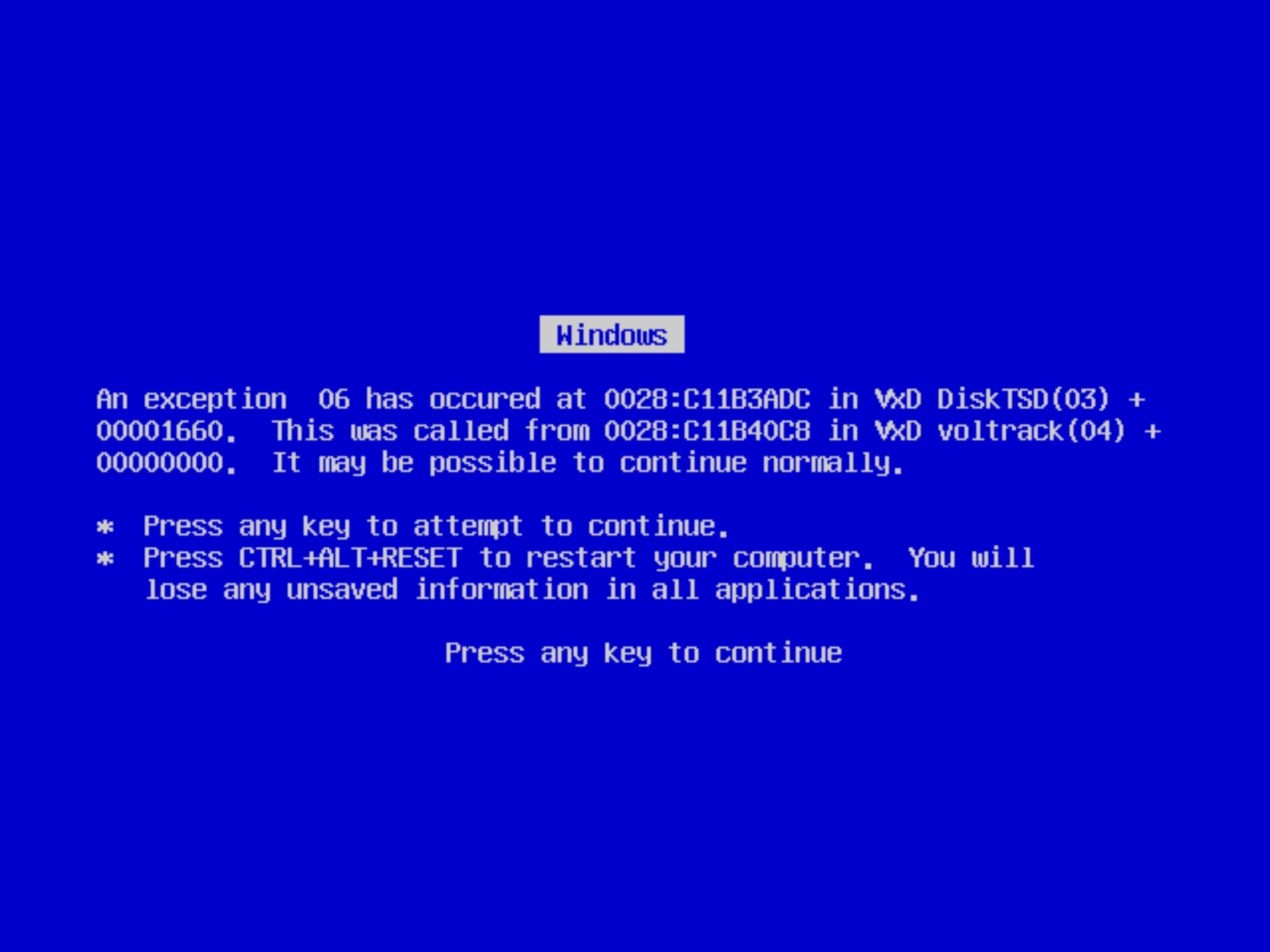
Gerwin Klein

Daniel Matichuk, June Andronick, Toby Murray, Mark Staples,

Ross Jefrey, Rafal Kolanski, Matthias Daum, Timothy Bourke

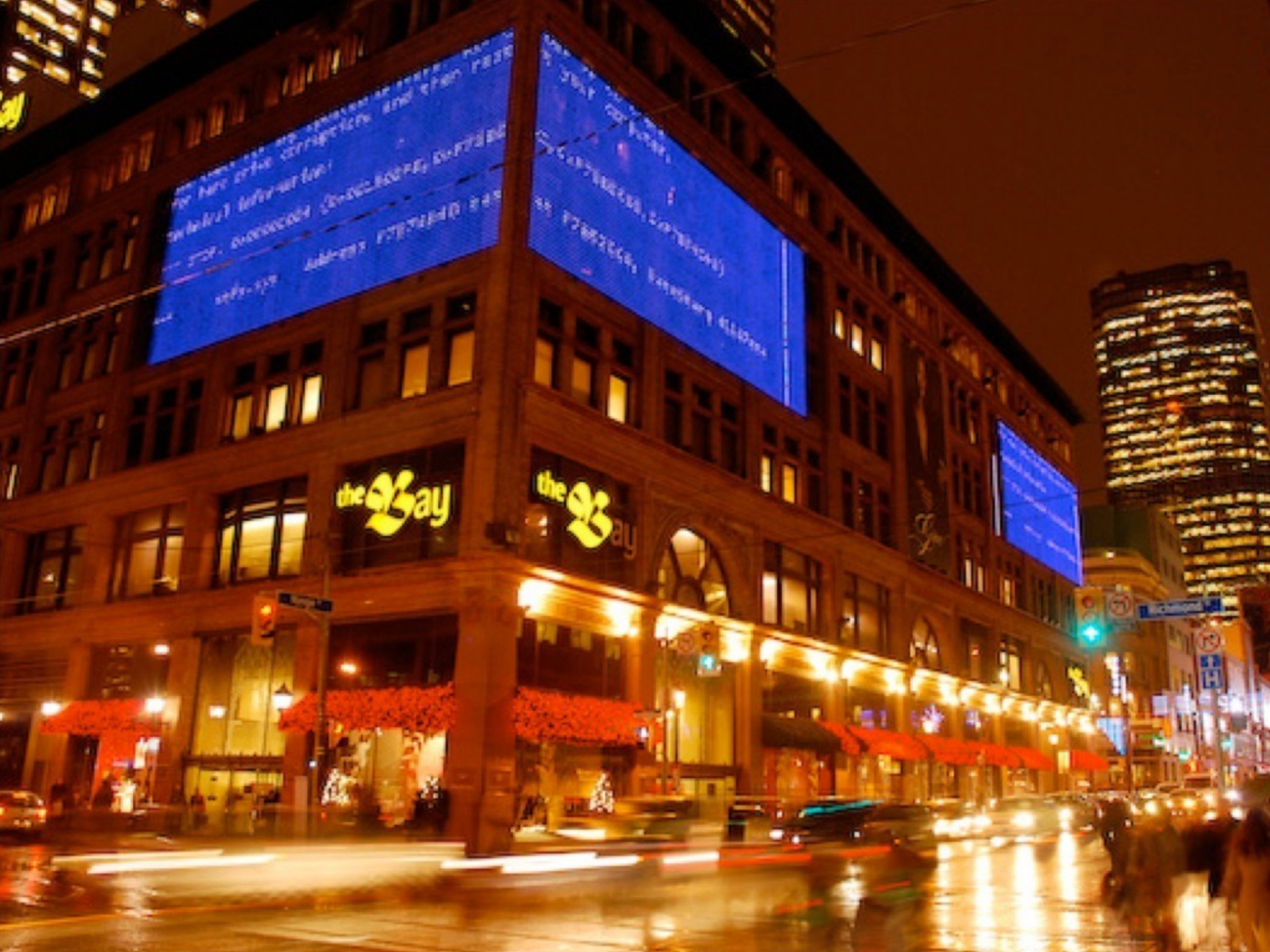
Australian Government
Department of Broadband, Communications and the Digital Economy
Australian Research Council

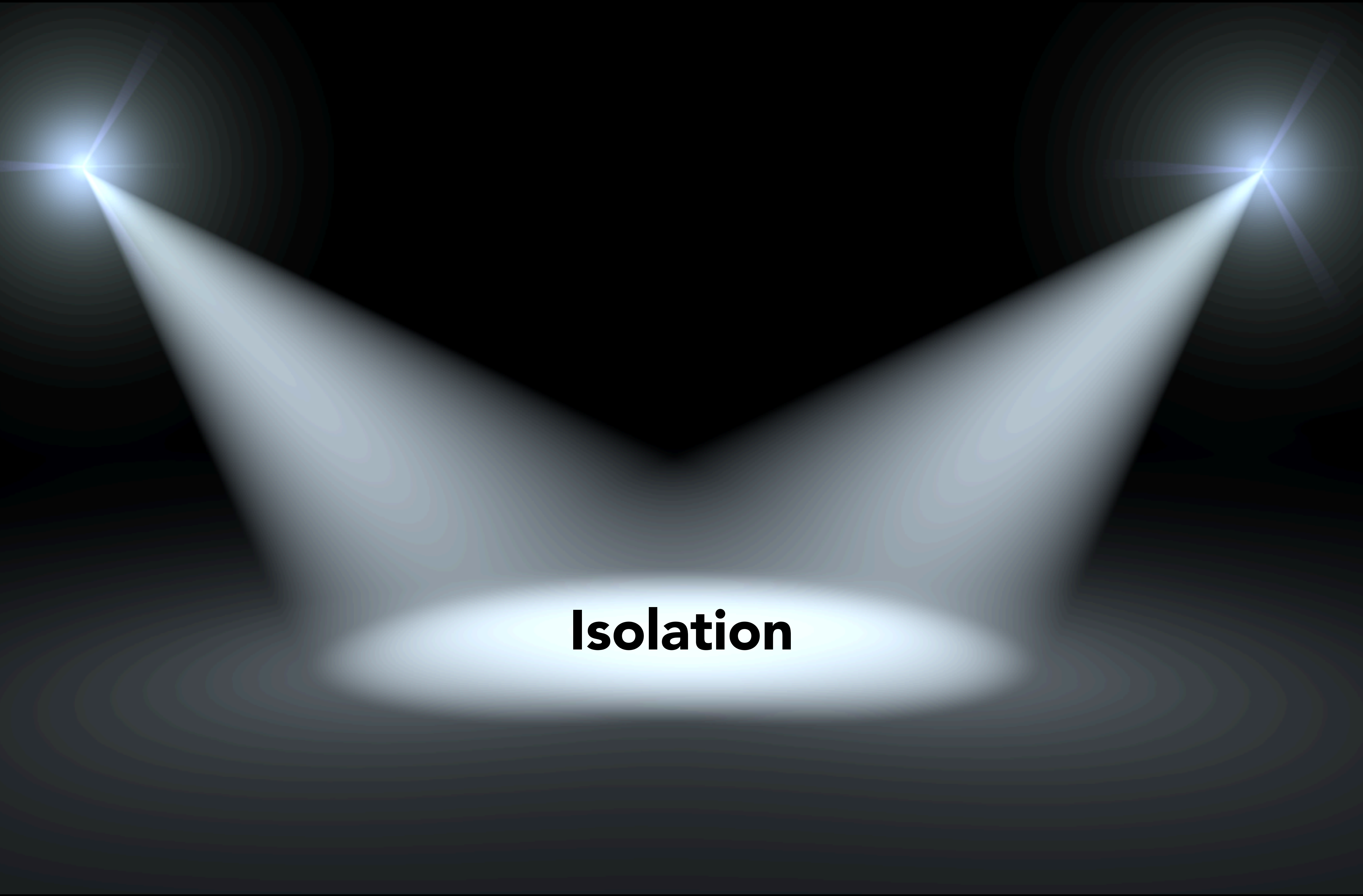
NICTA Funding and Supporting Members and Partners



```
                              Windows

An exception  06 has occured at 0028:C11B3ADC in VxD DiskTSD(03) +
00001660.  This was called from 0028:C11B40C8 in VxD voltrack(04) +
00000000.  It may be possible to continue normally.

*  Press any key to attempt to continue.
*  Press CTRL+ALT+RESET to restart your computer.  You will
   lose any unsaved information in all applications.

                      Press any key to continue
```

```
A problem has been detected and windows has been shut down to prevent damage
to your computer.

A process or thread crucial to system operation has unexpectedly exited or been
terminated.

If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:

Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any windows updates you might need.

If problems continue, disable or remove any newly installed hardware
or software. Disable BIOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restart
your computer, press F8 to select Advanced Startup options, and then
select Safe Mode.

Technical information:

*** STOP: 0x000000F4 (0x00000003,0x85866DA0,0x85866F14,0x805CBCA8)
```


the Bay
the Bay
Richmond



Isolation

Isolation is the Key

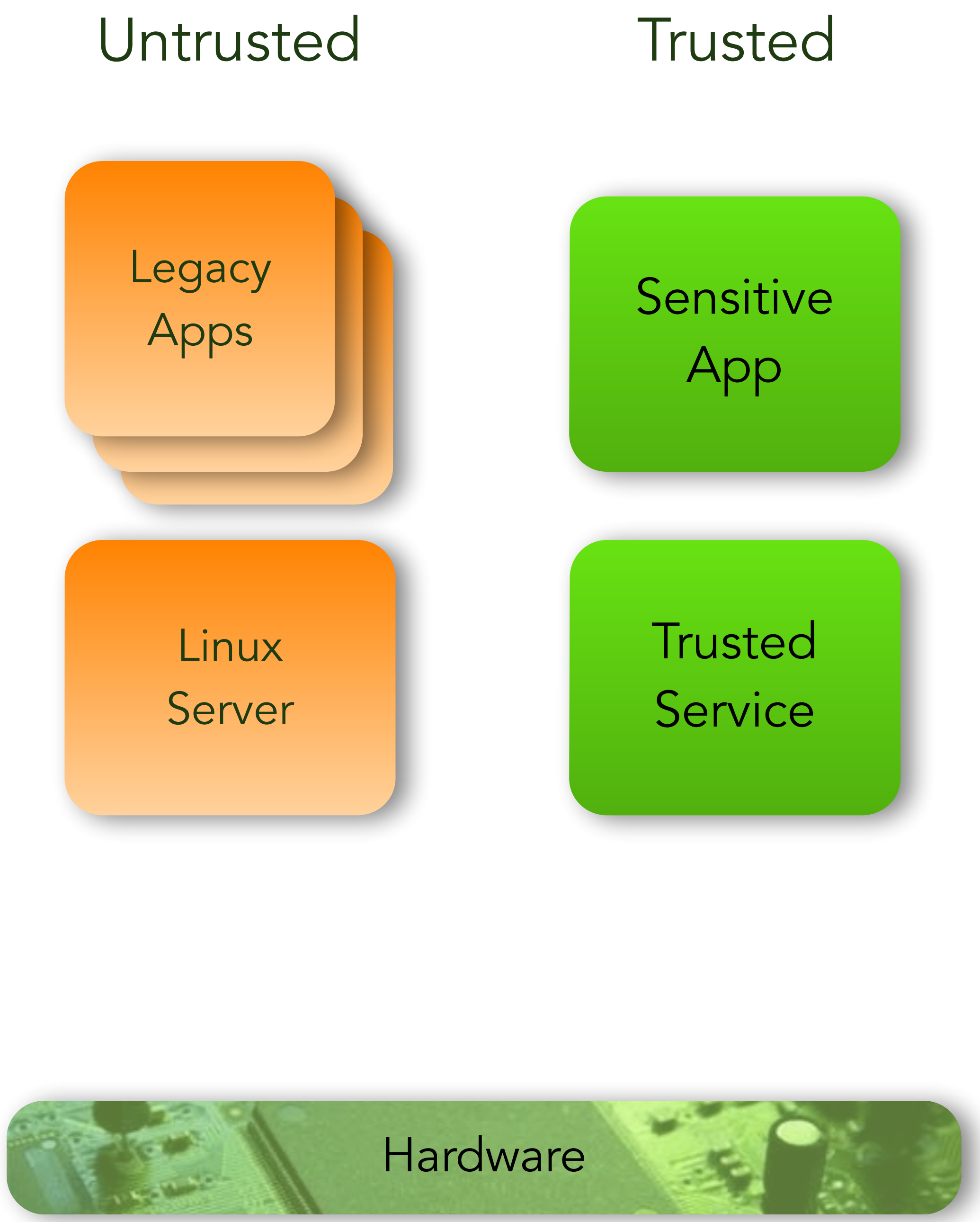
Trustworthy Computing Base

- message passing
- virtual memory
- interrupt handling
- access control

Applications

- fault isolation
- fault identification
- IP protection
- modularity

Trusted next to Untrusted



Isolation is the Key

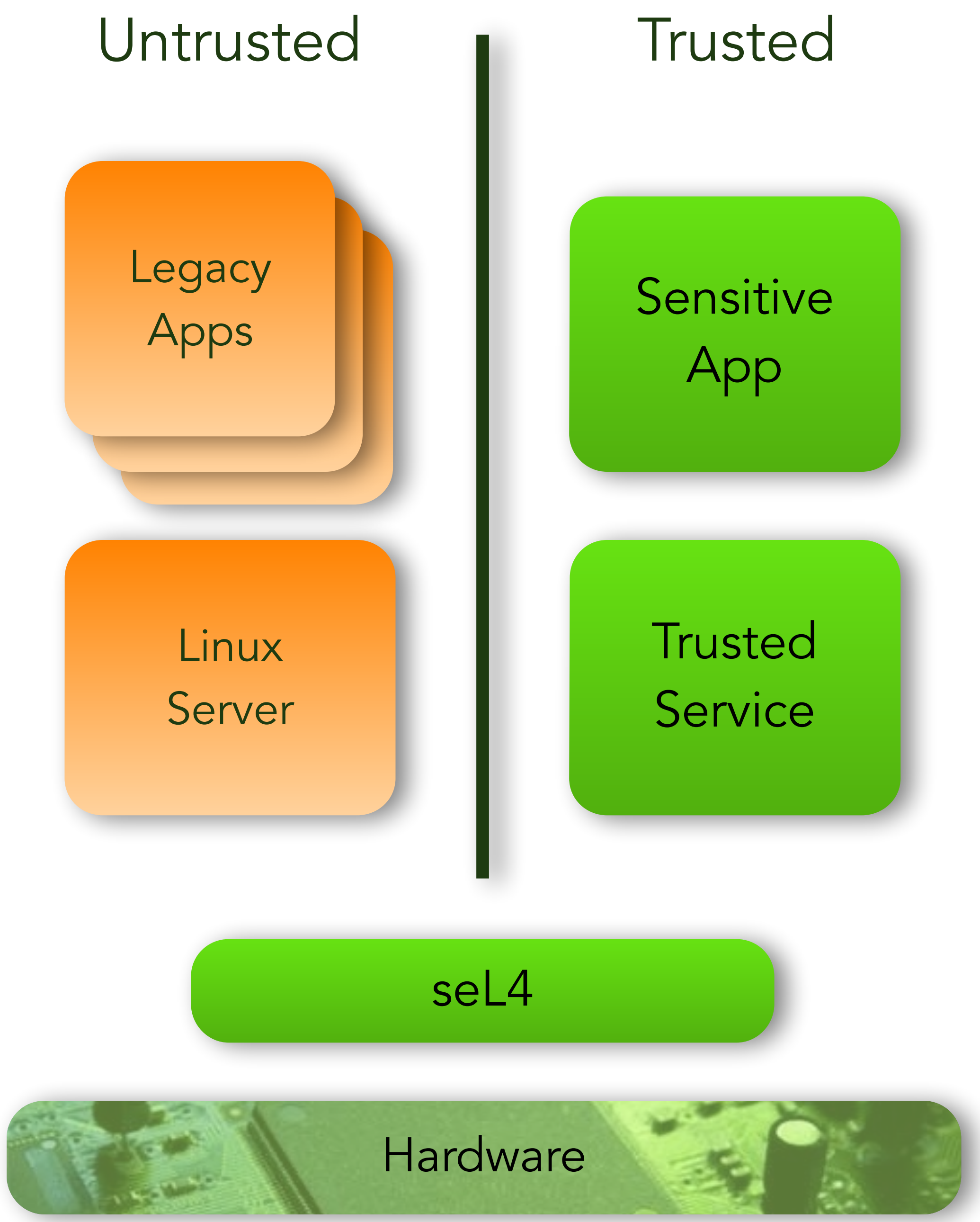
Trustworthy Computing Base

- message passing
- virtual memory
- interrupt handling
- access control

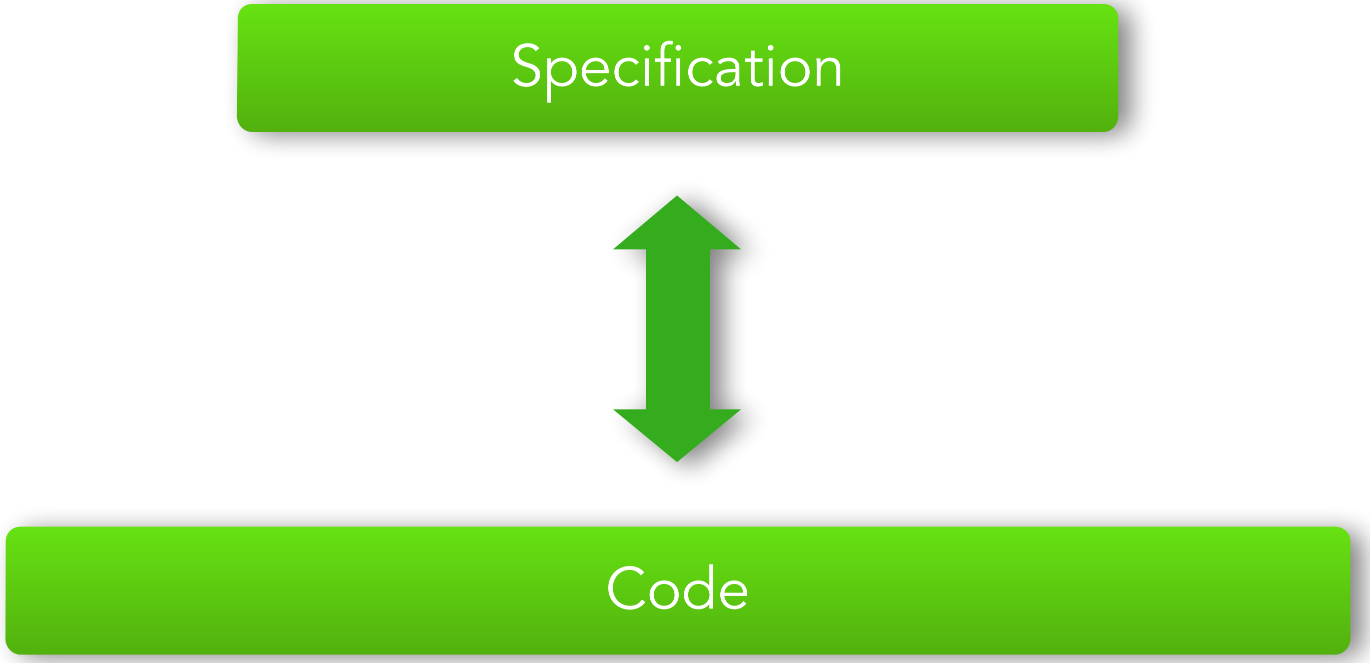
Applications

- fault isolation
- fault identification
- IP protection
- modularity

Trusted next to Untrusted



Functional Correctness Possible



Specification

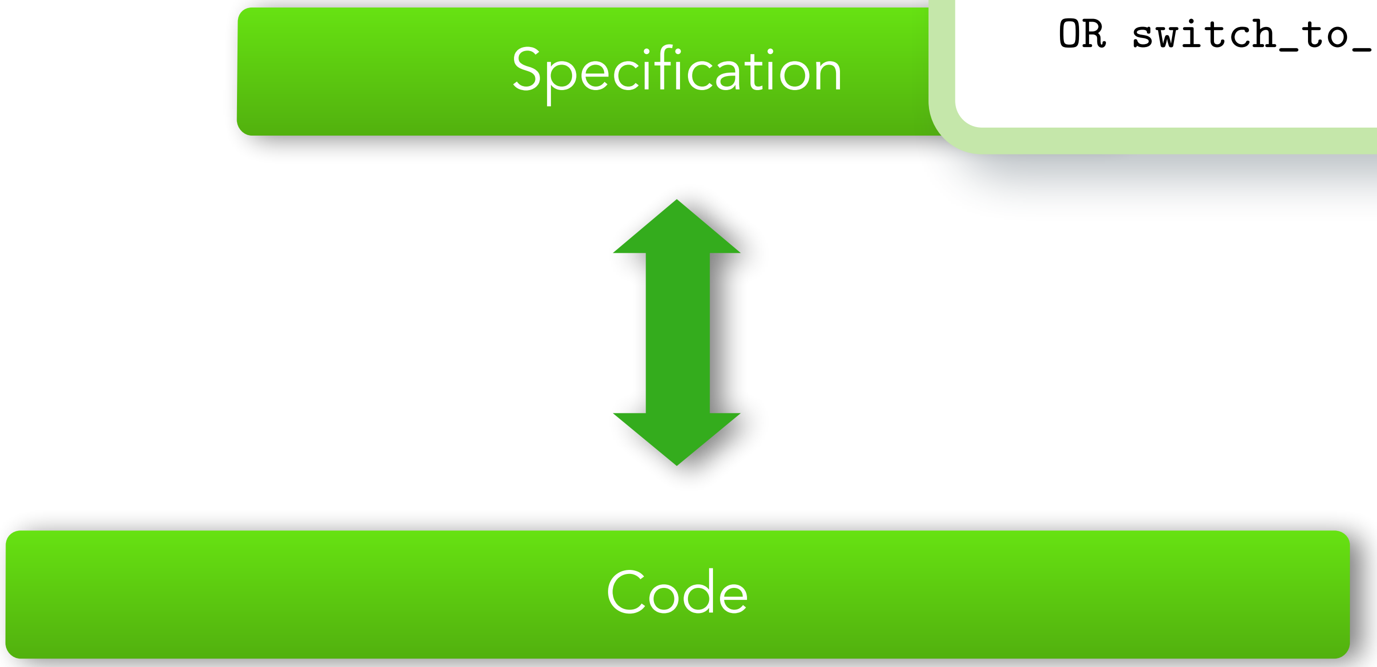
Proof

Code

Functional Correctness Possible

What

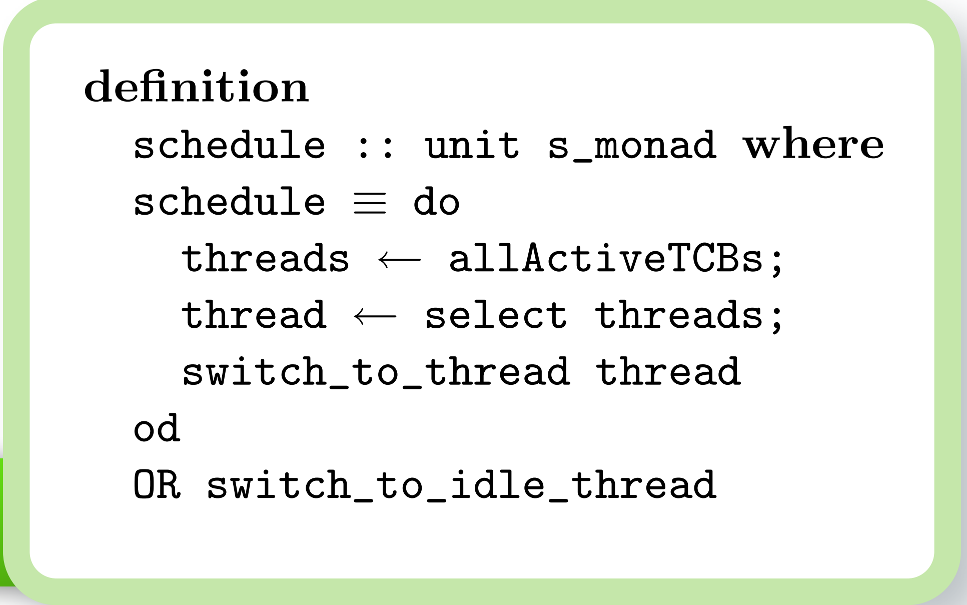
Proof



```
definition
  schedule :: unit s_monad where
  schedule ≡ do
    threads ← allActiveTCBs;
    thread ← select threads;
    switch_to_thread thread
  od
  OR switch_to_idle_thread
```

Functional Correctness Possible

What

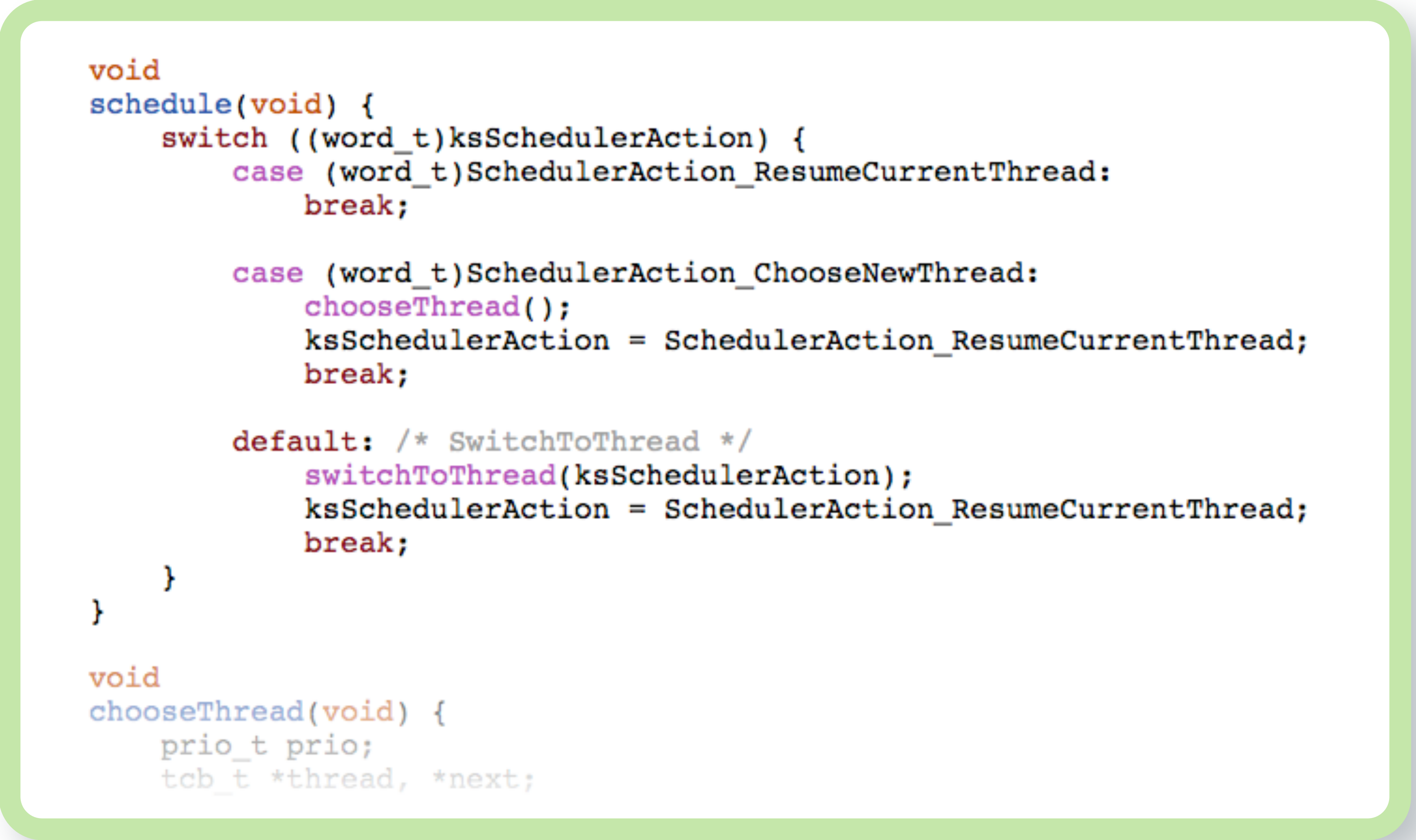


```
definition
  schedule :: unit s_monad where
  schedule ≡ do
    threads ← allActiveTCBs;
    thread ← select threads;
    switch_to_thread thread
  od
  OR switch_to_idle_thread
```

Specification

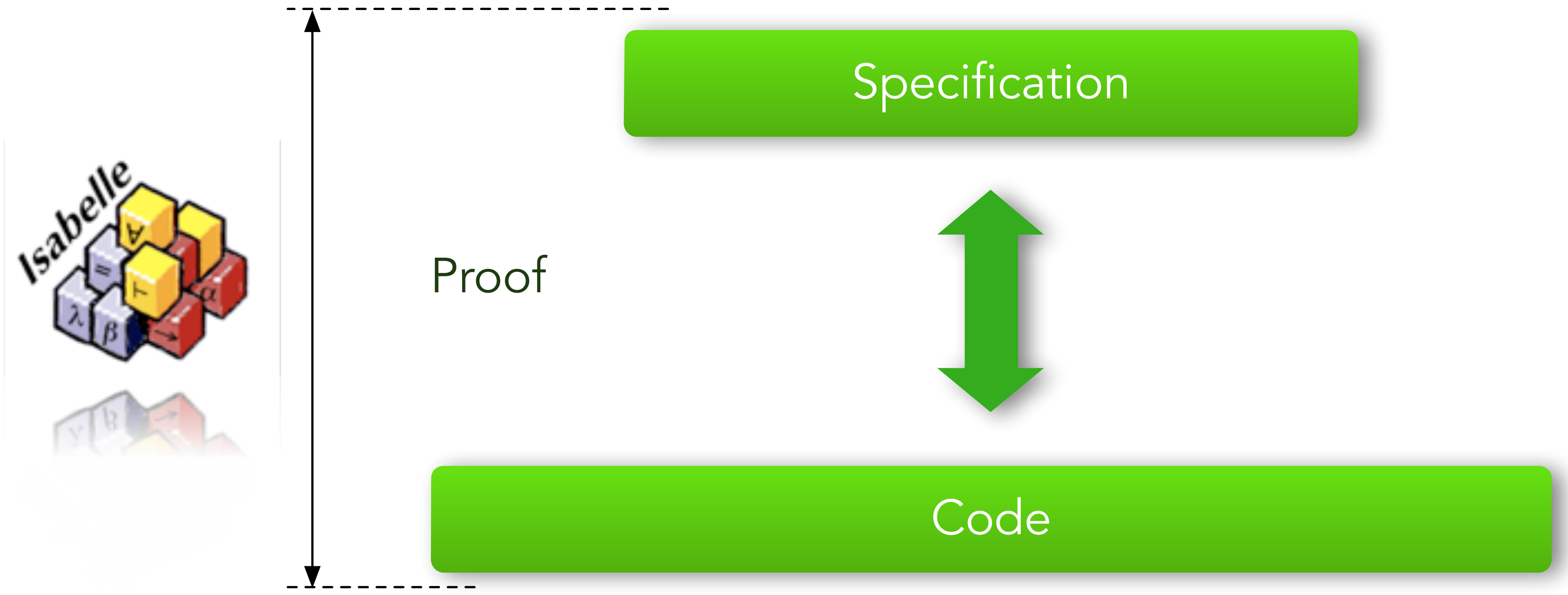
Proof

How



```
void
schedule(void) {
    switch ((word_t)ksSchedulerAction) {
        case (word_t)SchedulerAction_ResumeCurrentThread:
            break;

        case (word_t)SchedulerAction_ChooseNewThread:
            chooseThread();
            ksSchedulerAction = SchedulerAction_ResumeCurrentThread;
            break;

        default: /* SwitchToThread */
            switchToThread(ksSchedulerAction);
            ksSchedulerAction = SchedulerAction_ResumeCurrentThread;
            break;
    }
}

void
chooseThread(void) {
    prio_t prio;
    tcb_t *thread, *next;
```

*conditions apply

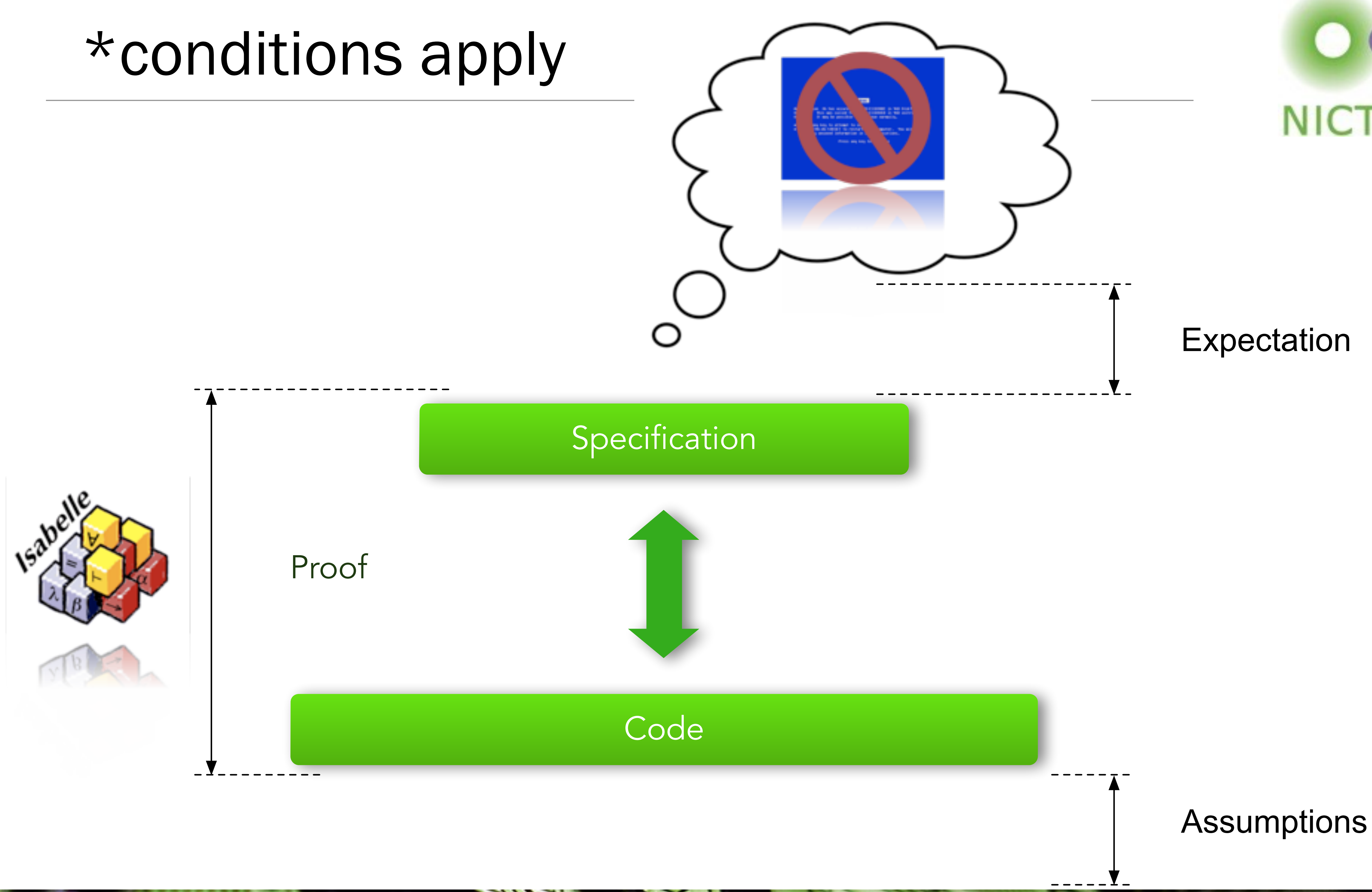


Specification

Proof

Code

*conditions apply



Expectation

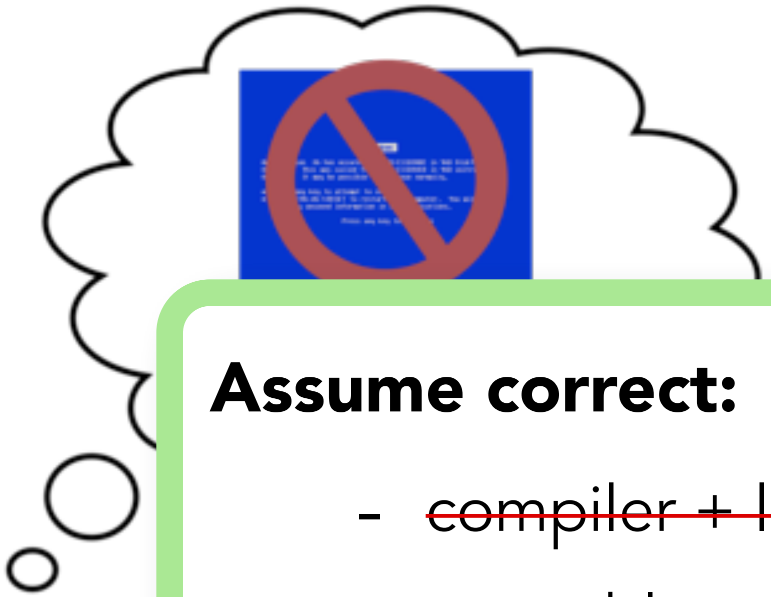
Specification

Proof

Code

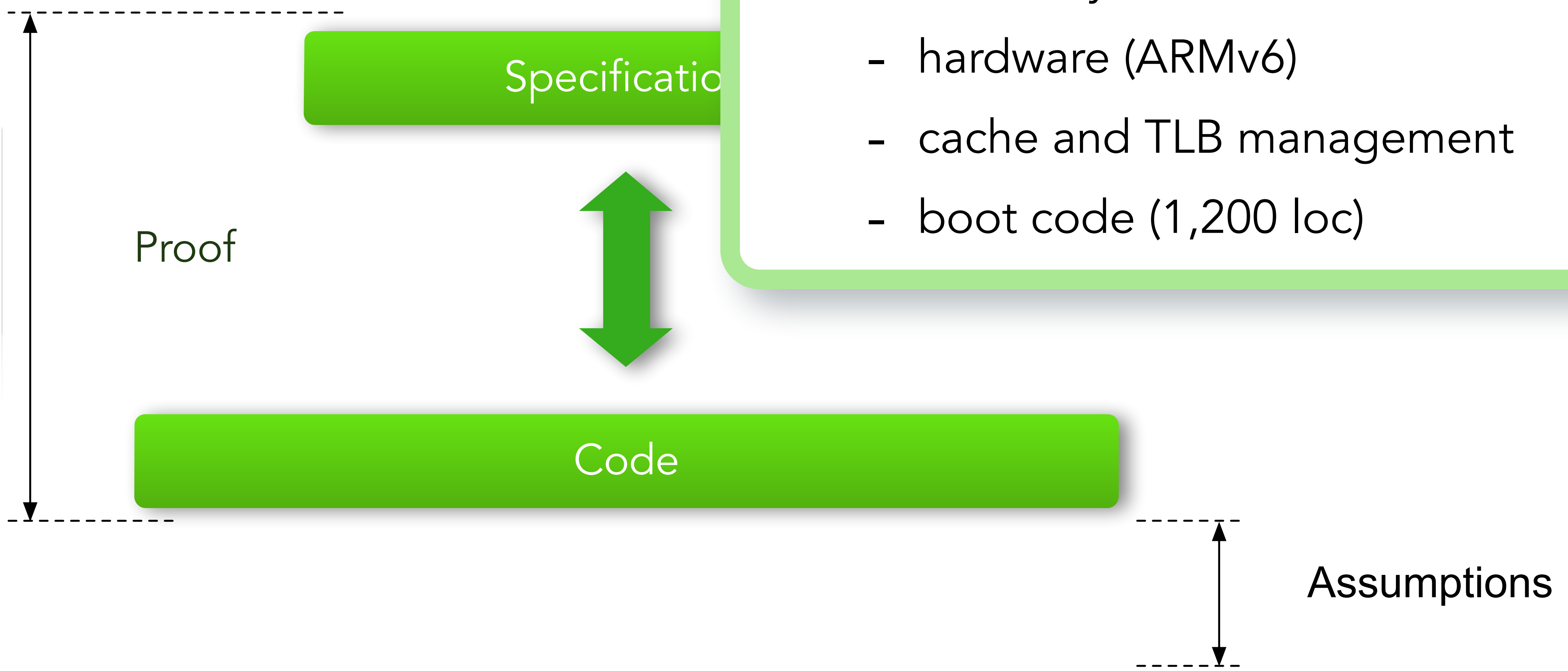
Assumptions

*conditions apply

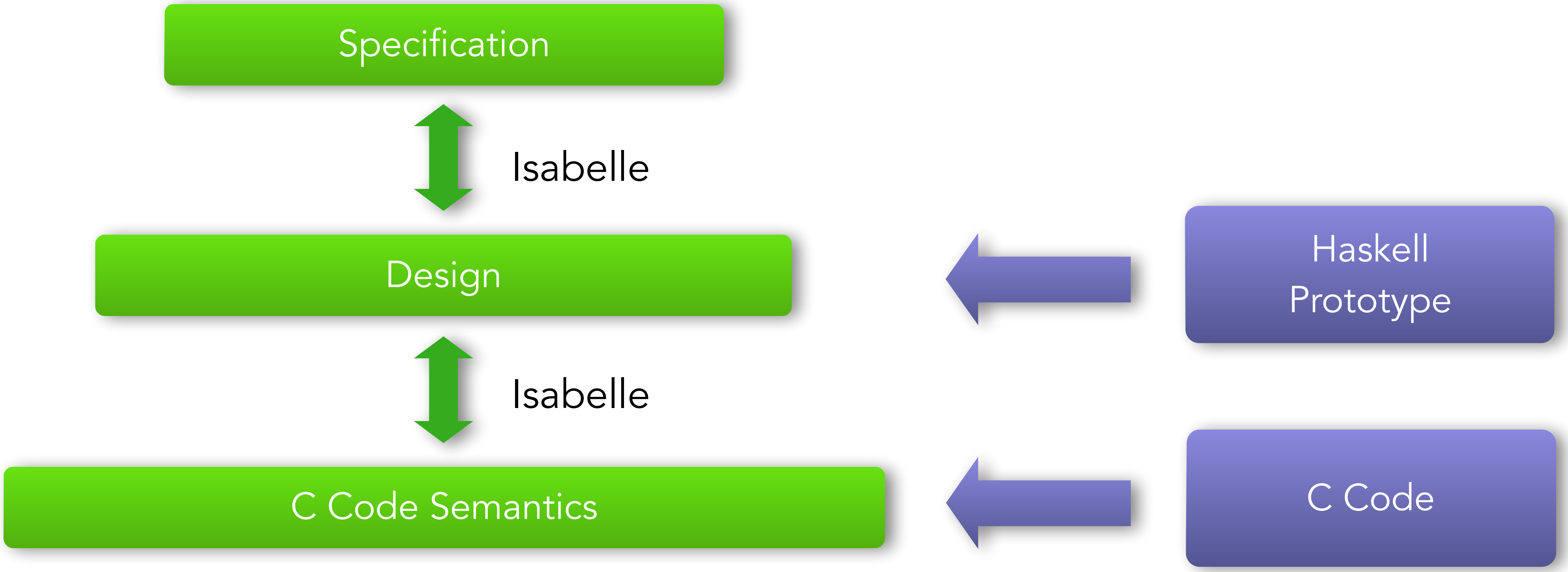


Assume correct:

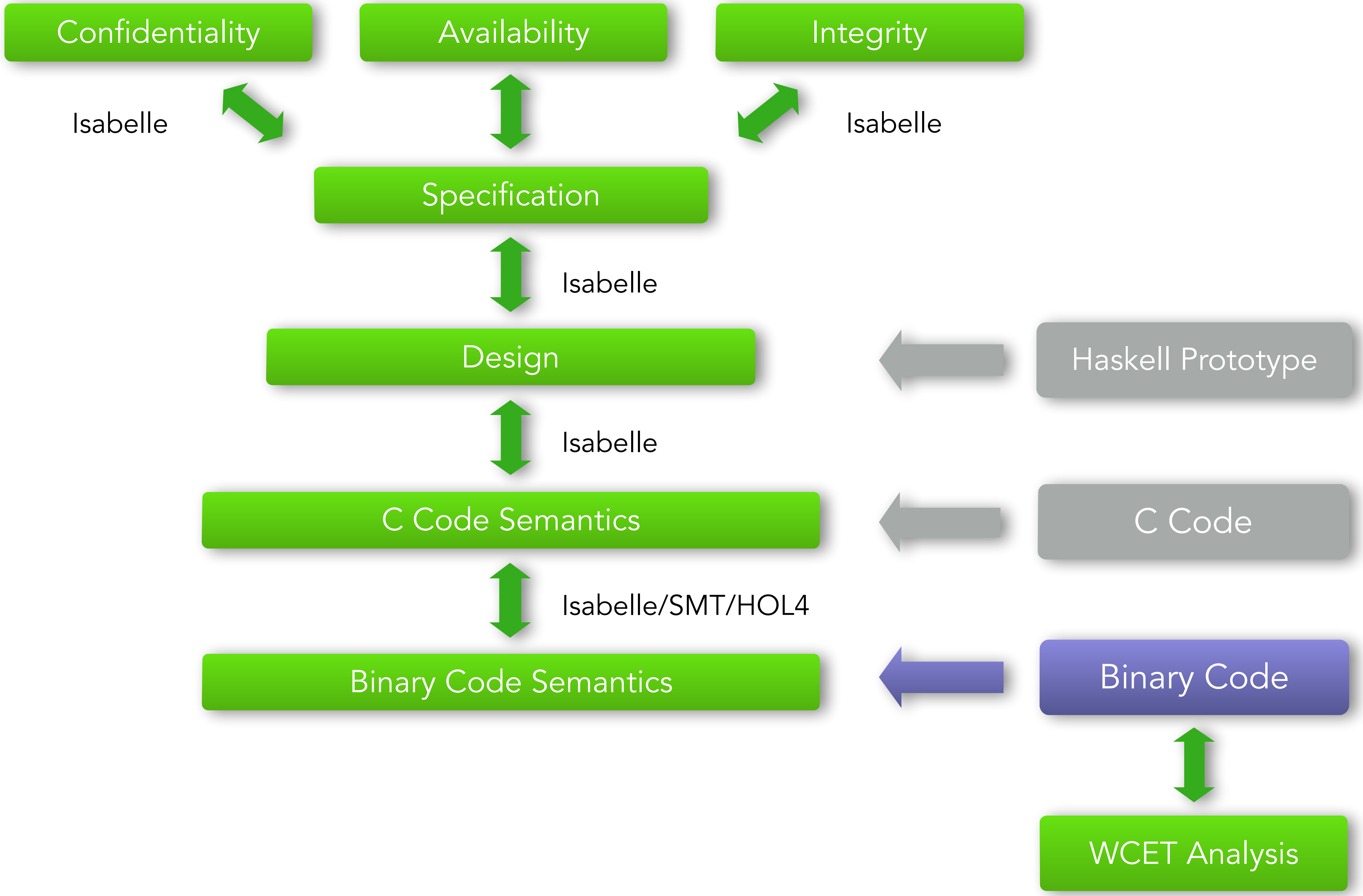
- ~~compiler + linker (wrt. C op sem)~~
- assembly code (600 loc)
- hardware (ARMv6)
- cache and TLB management
- boot code (1,200 loc)



Proof Architecture [SOSP'09]



Proof Architecture Now



Confidentiality

Availability

Integrity

Isabelle

Isabelle

Specification

Isabelle

Design

Haskell Prototype

Isabelle

C Code Semantics

C Code

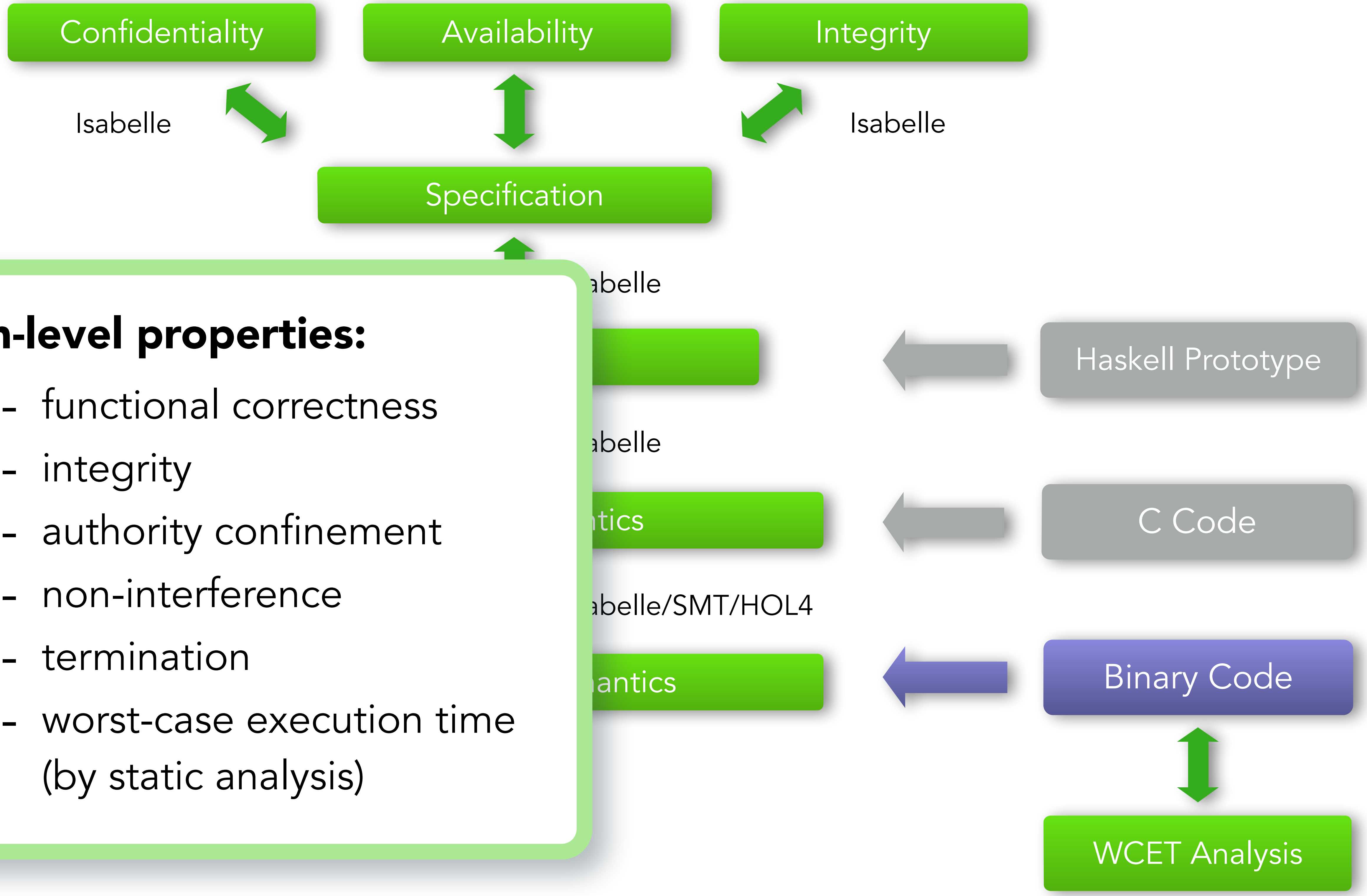
Isabelle/SMT/HOL4

Binary Code Semantics

Binary Code

WCET Analysis

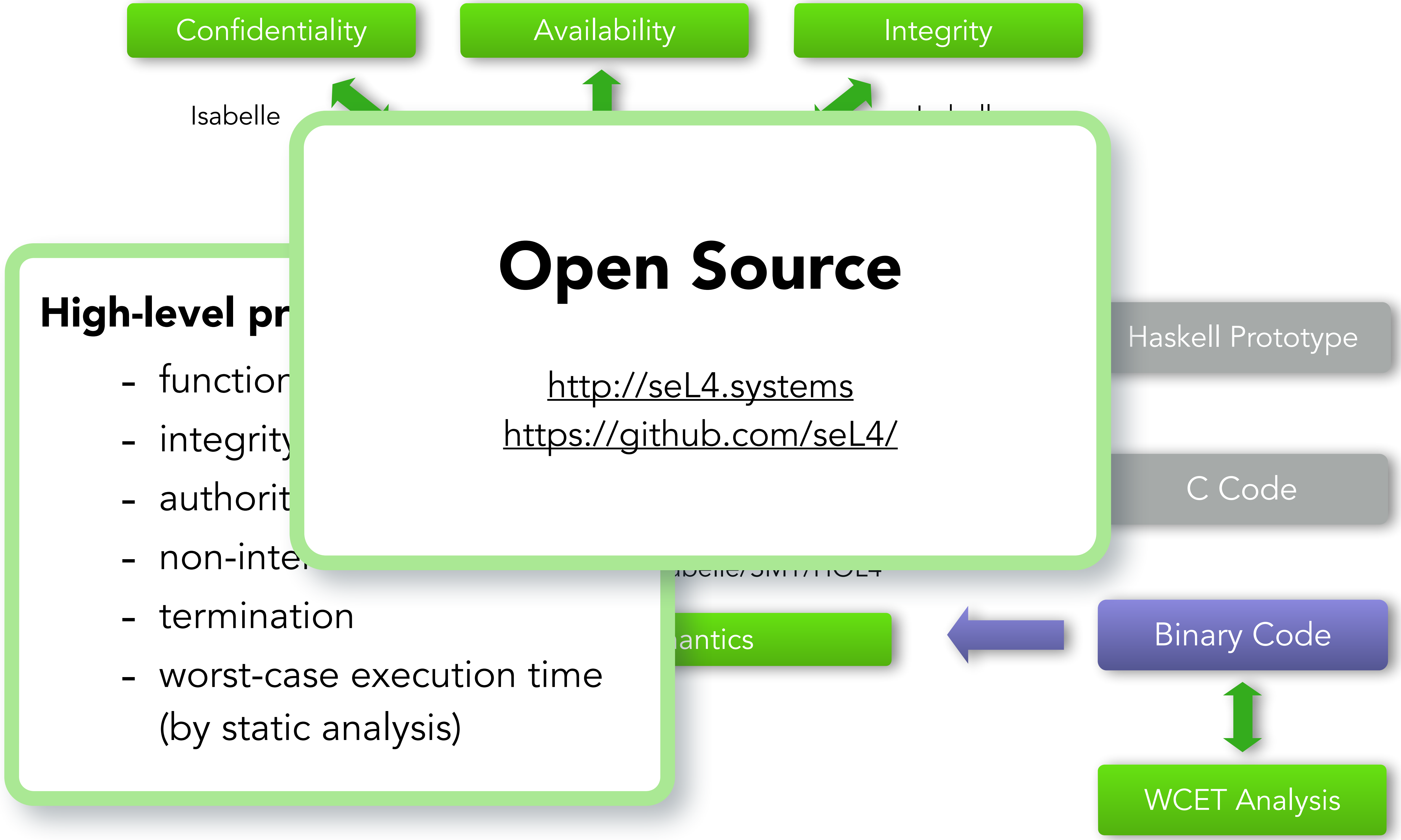
Proof Architecture Now



High-level properties:

- functional correctness
- integrity
- authority confinement
- non-interference
- termination
- worst-case execution time (by static analysis)

Proof Architecture Now



Open Source

http://seL4.systems
https://github.com/seL4/

High-level pr

- function
- integrity
- authorit
- non-inte
- termination
- worst-case execution time (by static analysis)

Next Step: Full System Assurance

DARPA HACMS Program:

- Provable vehicle safety
- Red Team must not be able to divert vehicle

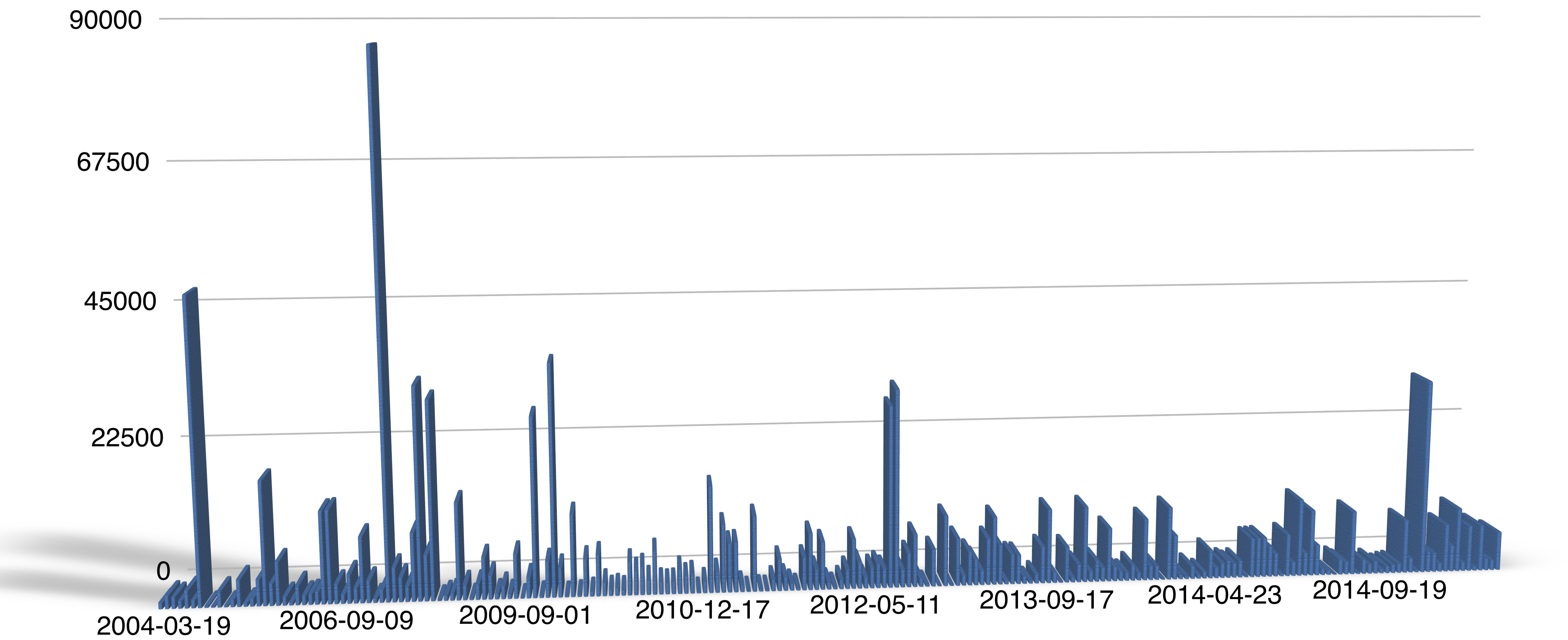
Boeing Unmanned Little Bird (AH-6)

SMACCMcopter Research Vehicle

Scale

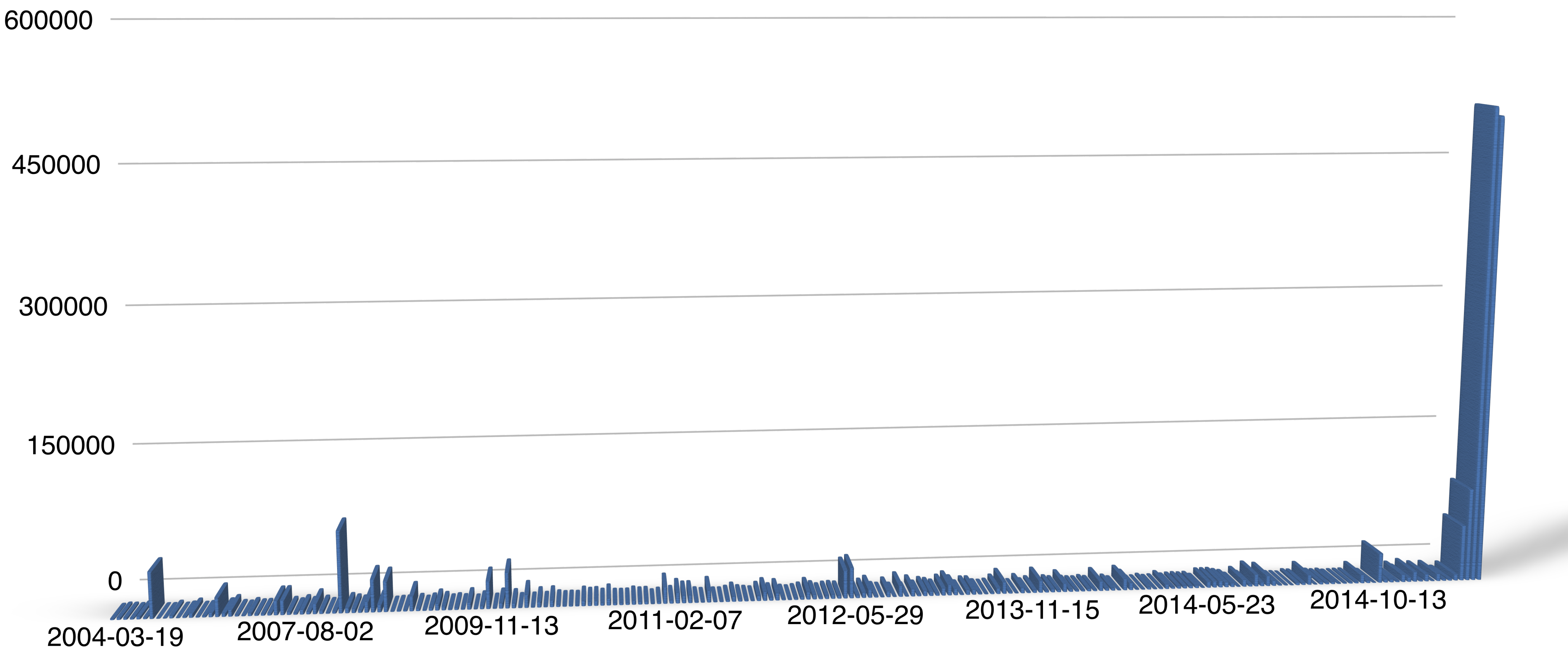
Scale

Scale



size of AFP entries by submission date

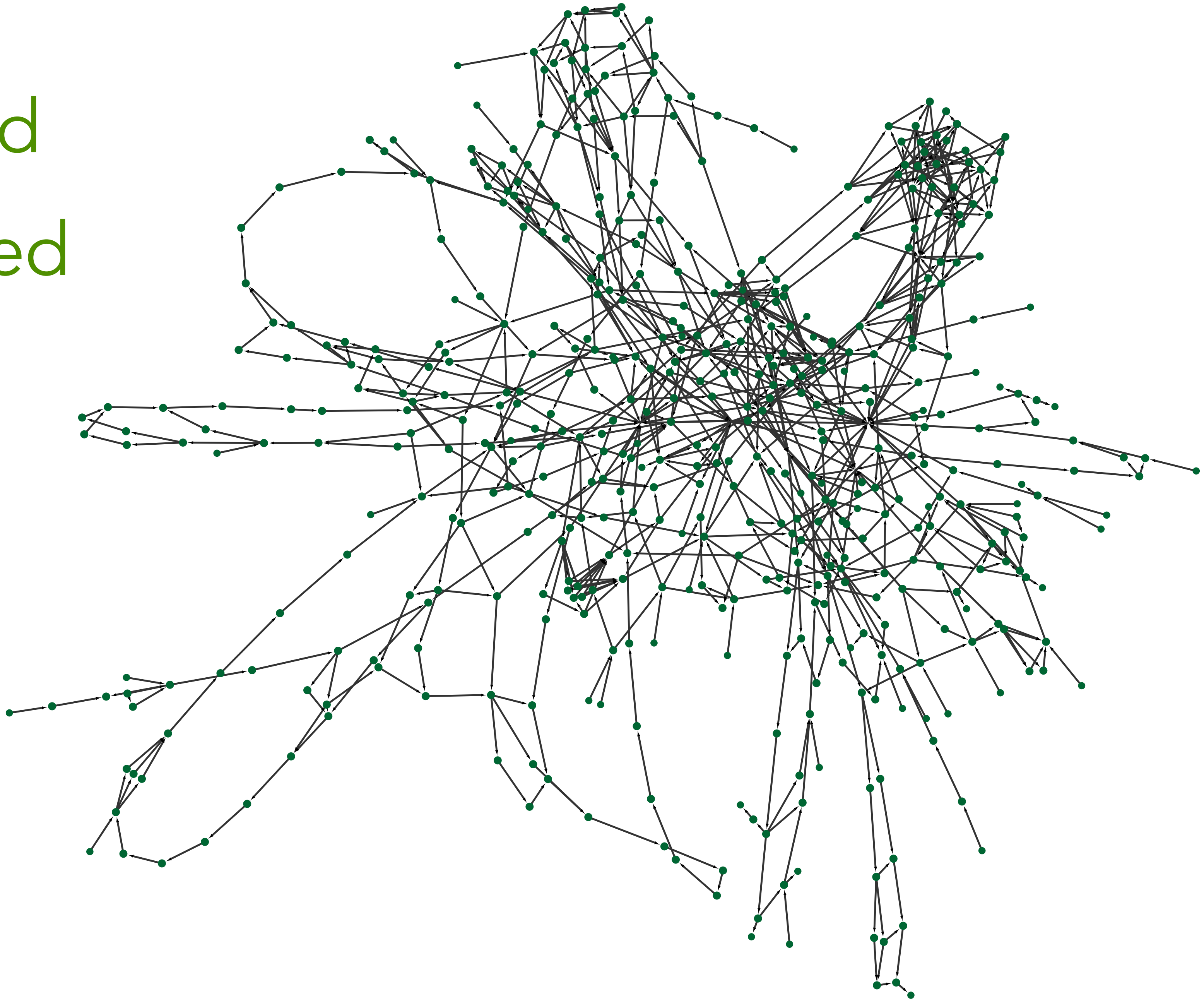
Scale



size of AFP entries by submission date
with four-colour theorem, odd-order theorem, Verisoft, L4,verified

Proof Introspection

- 500 files
- 22,000 lemmas stated
- 95,000 lemmas proved



Proof Introspection

- 500 files
- 22,000 lemmas stated
- 95,000 lemmas proved

Raf's Observation

The introspection of proof and theories is an essential part of working on a large-scale verification development.

Proof Introspection

- 500 files
- 22,000 lemmas stated
- 95,000 lemmas proved

Raf's Observation

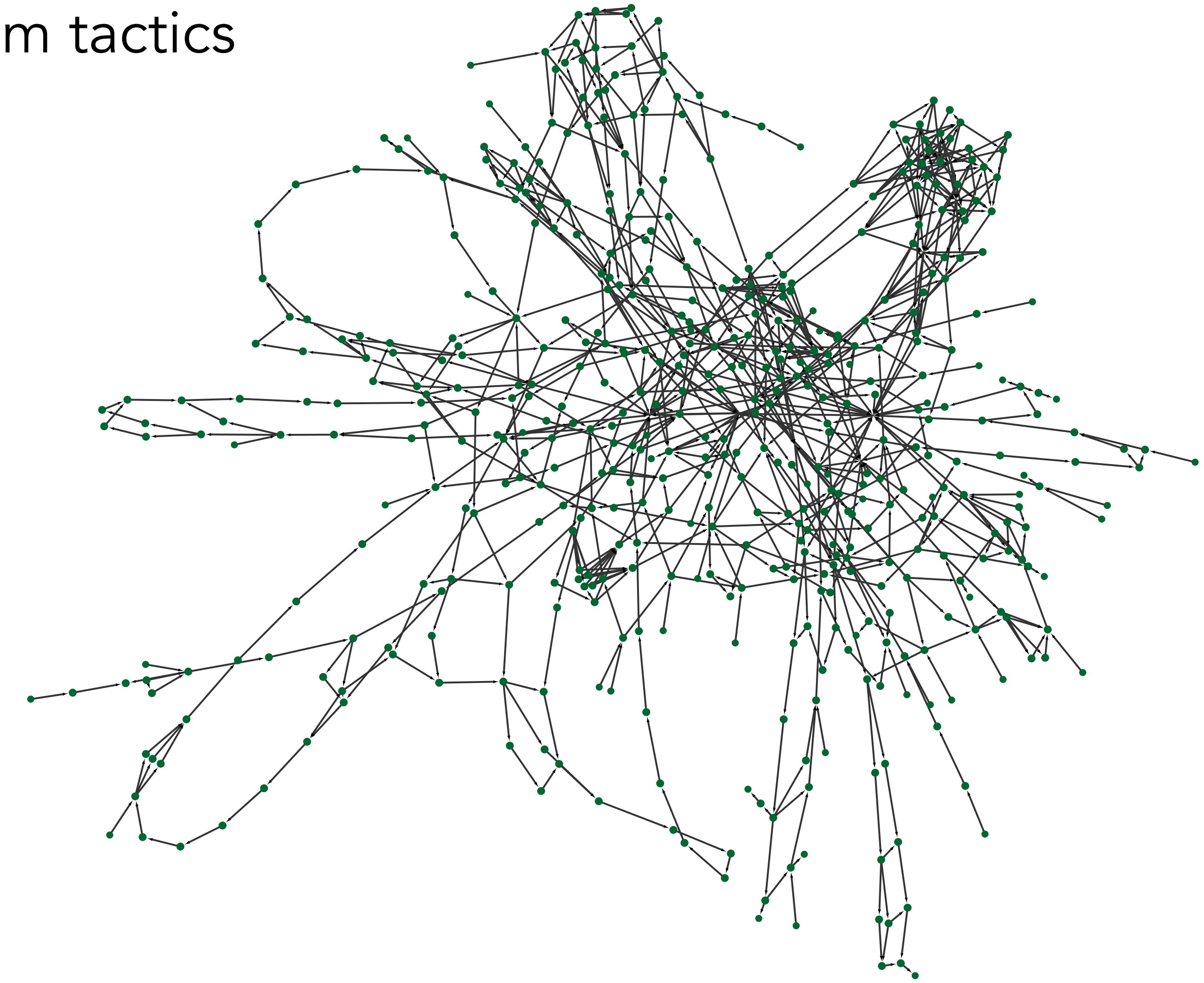
The introspection of proof and theories is an essential part of working on a large-scale verification development.

- Learning Isabelle? **Easy.**
- Learning microkernels? **Not too bad.**
- Finding your way in the 500kloc proof jungle? **Hard!**

Proof Development

– proof development

- decomposition of proofs over people,
- custom proof calculus,
- automating mechanical tasks, custom tactics
- proof craft



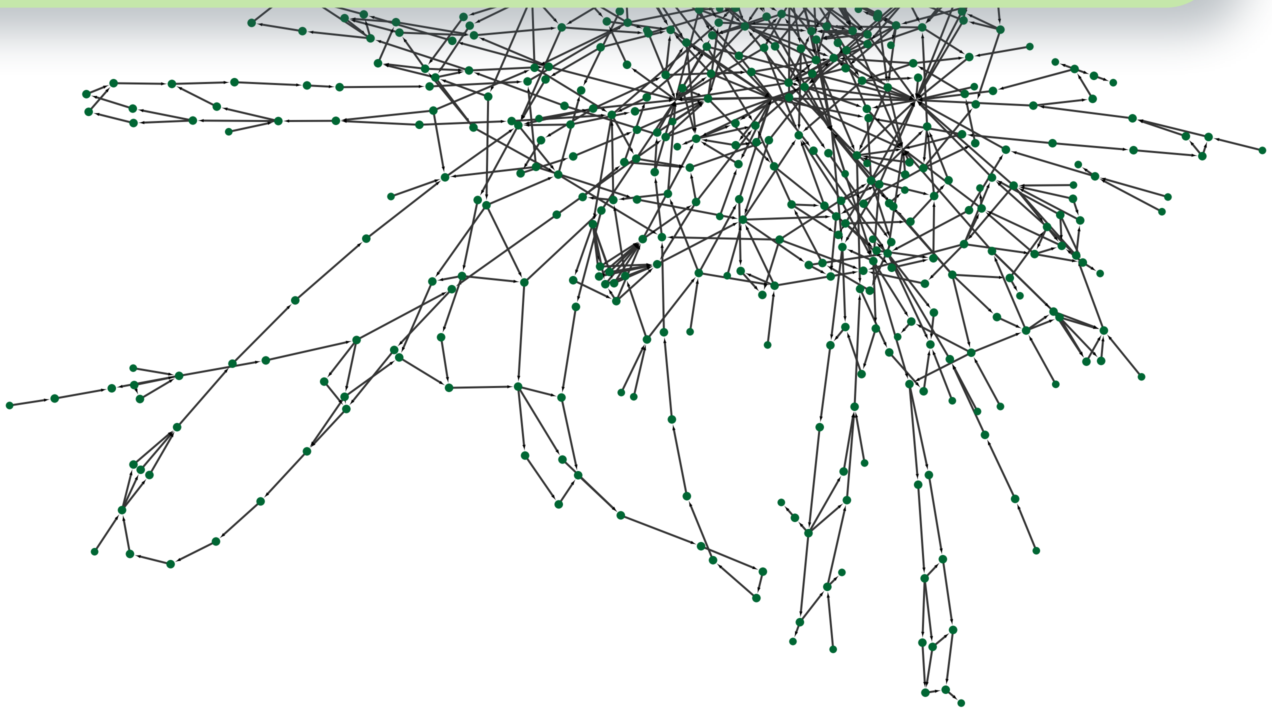
Proof Development

– proof development

- decomposition of proofs ov
- custom proof calculus,
- automating mechanical task
- proof craft

Tim's Statement

Automating "donkey work" allows attention and effort to be focussed where most needed – but it must be done judiciously.



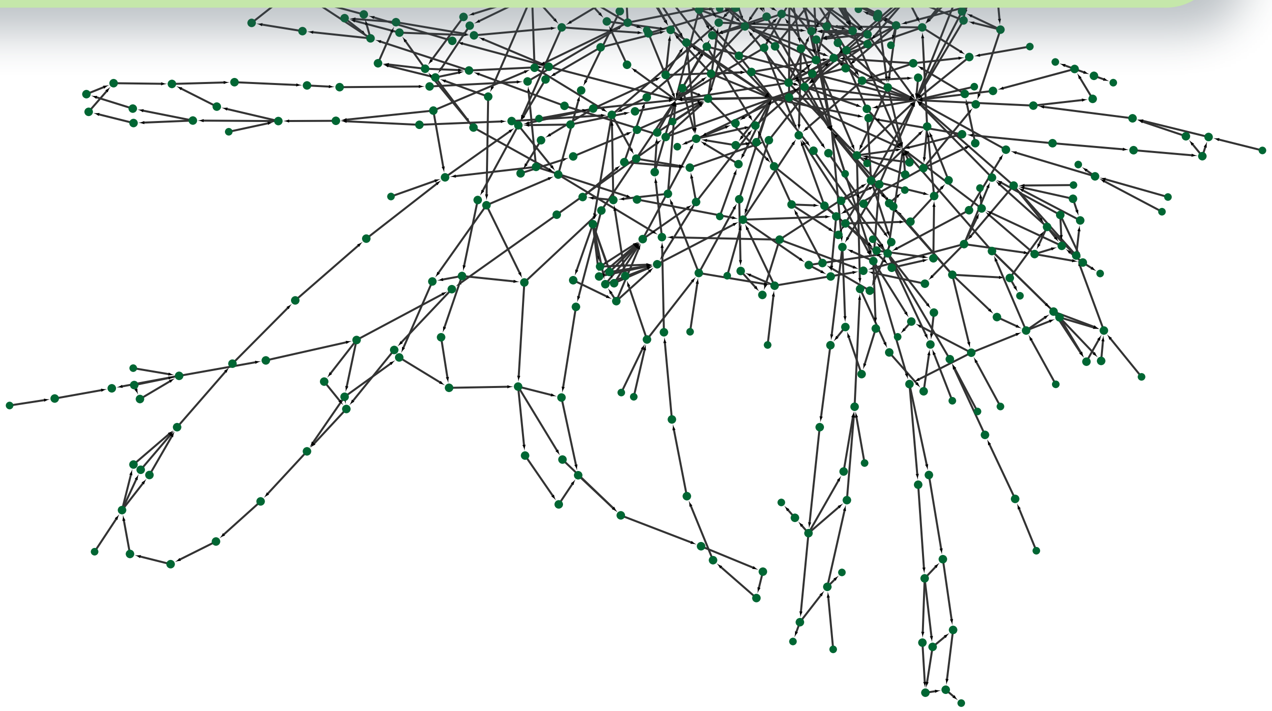
Proof Development

– proof development

- decomposition of proofs ov
- custom proof calculus,
- automating mechanical task
- proof craft

Tim's Statement

Automating "donkey work" allows attention and effort to be focussed where most needed – but it must be done judiciously.



– challenges

- non-local change,
- speculative change,
- distributed development

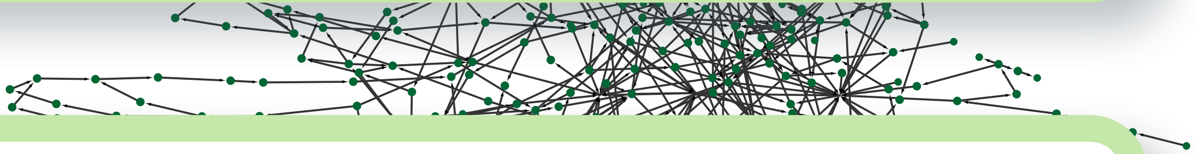
Proof Development

– proof development

- decomposition of proofs ov
- custom proof calculus,
- automating mechanical tas
- proof craft

Tim's Statement

Automating "donkey work" allows attention and effort to be focussed where most needed – but it must be done judiciously.



– challenges

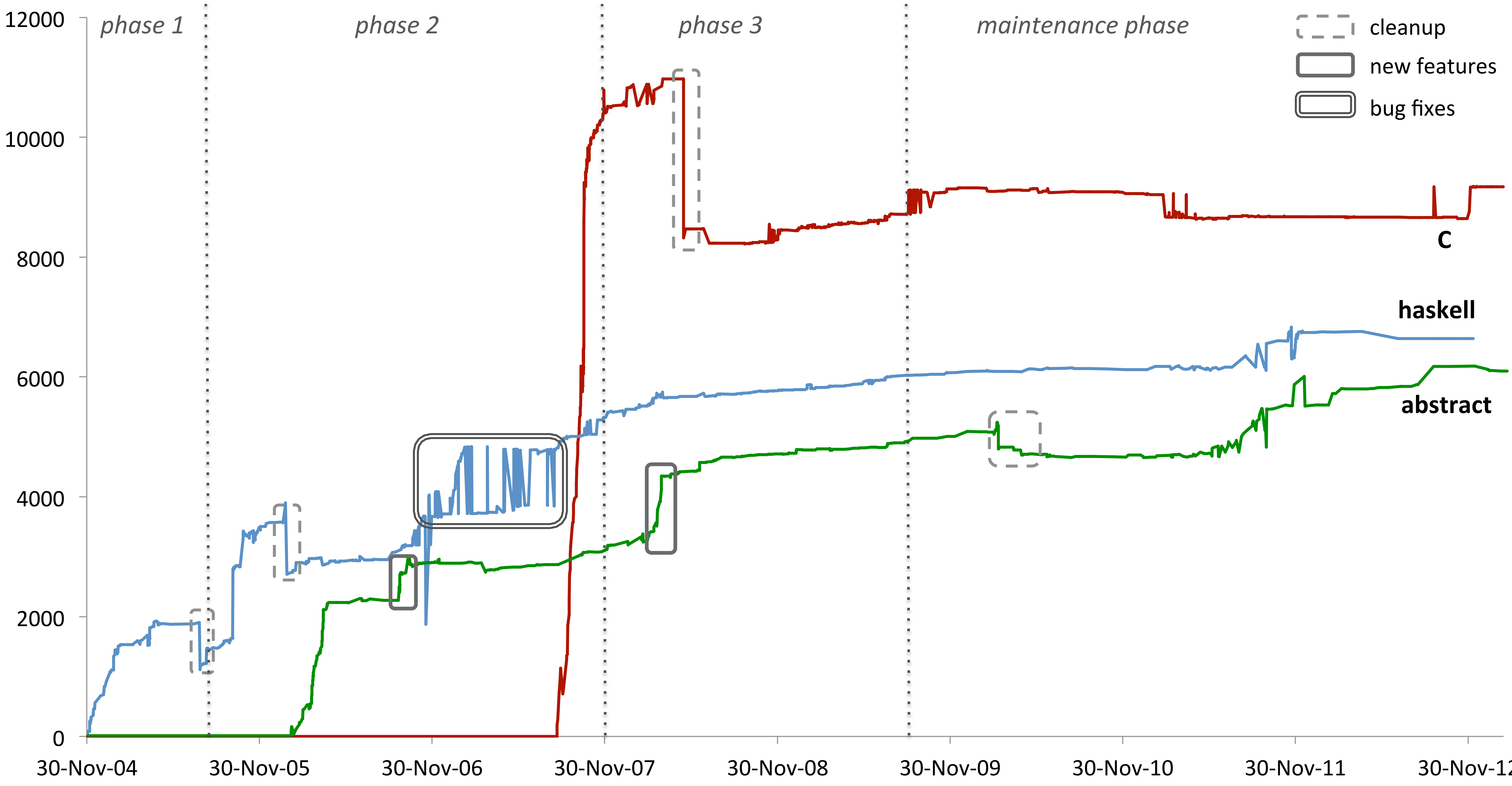
- non-local change,
- speculative change,
- distributed development

Matthias' Conjecture

Over the years, I must have waited weeks for Isabelle. Productivity hinges on a short edit-check cycle; for that, I am even willing to (temporarily) sacrifice soundness.

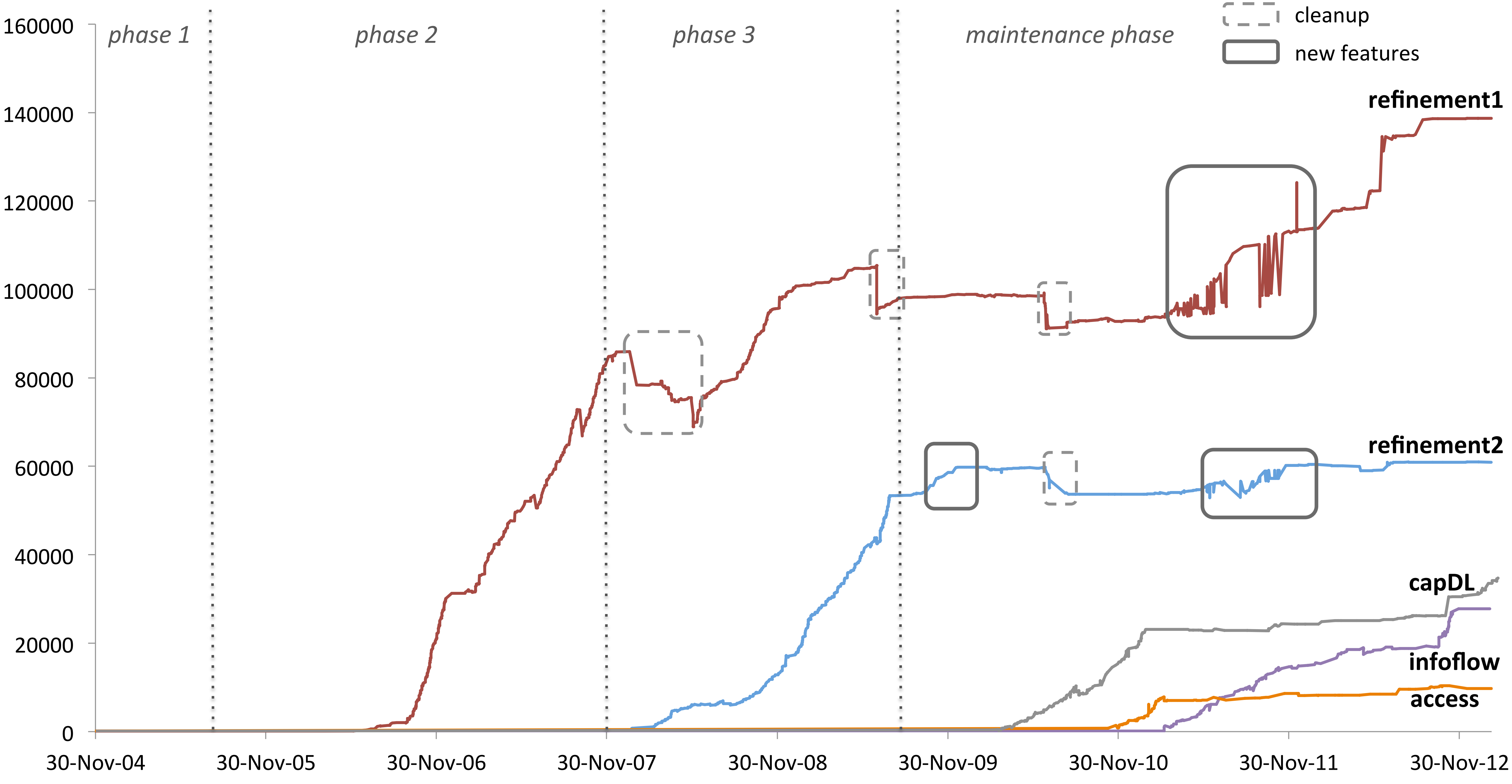
Maintenance

- Development of seL4 code + spec artefacts (sloc)



Maintenance

- Development of seL4 proofs (sloc)



(b) Size of proofs (X: time; Y: SLOC)

Problems of Scale

– proof maintenance

- changes, updates, new proofs, new features
- automated regression, keep code in sync
- refactoring
- simplification

Problems of Scale

– proof maintenance

- changes, updates, new proofs, new features
- automated regression, keep code in sync
- refactoring
- simplification

Dan's Conclusion

Verification is fast, maintenance is forever.

Research Challenges

Software vs Proof Engineering

- Is Proof Engineering a thing?
  - Google Scholar:
    - “software engineering”   1,430,000 results

Software vs Proof Engineering

- Is Proof Engineering a thing?
  - Google Scholar:

“software engineering”	1,430,000 results
“proof engineering”	564 results

Software vs Proof Engineering

- Is Proof Engineering a thing?
  - Google Scholar:

“software engineering”	1,430,000 results
“proof engineering”	564 results

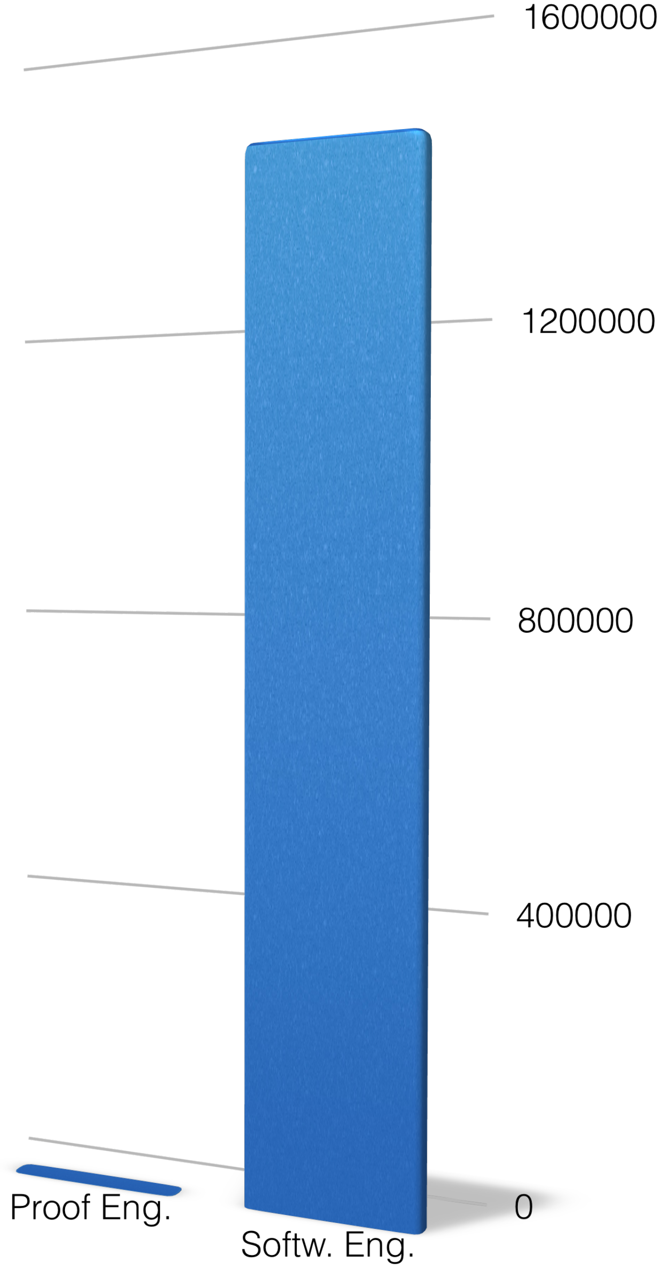
Includes

”The Fireproof Building” and

”Influence of water permeation and analysis of treatment for the Longmen Grottoes"

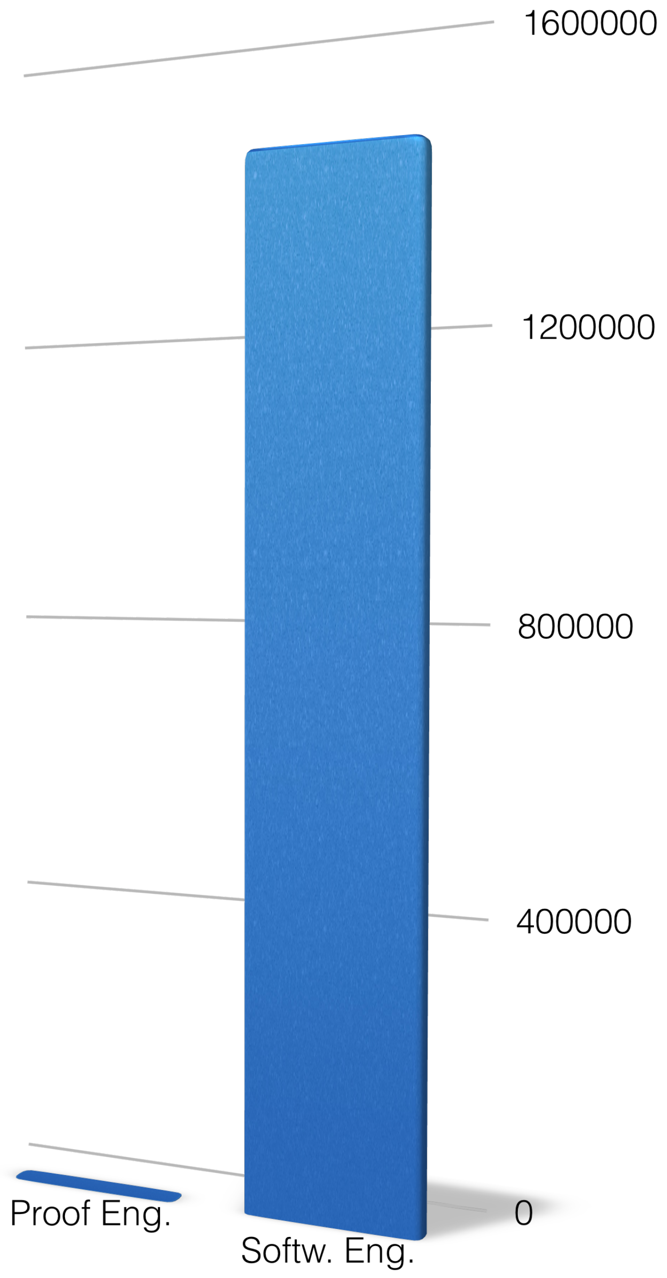
Proof Engineering is The Same

- Same kind of artefacts:
  - lemmas are functions, modules are modules
  - code gets big too
  - version control, regressions, refactoring and IDEs apply



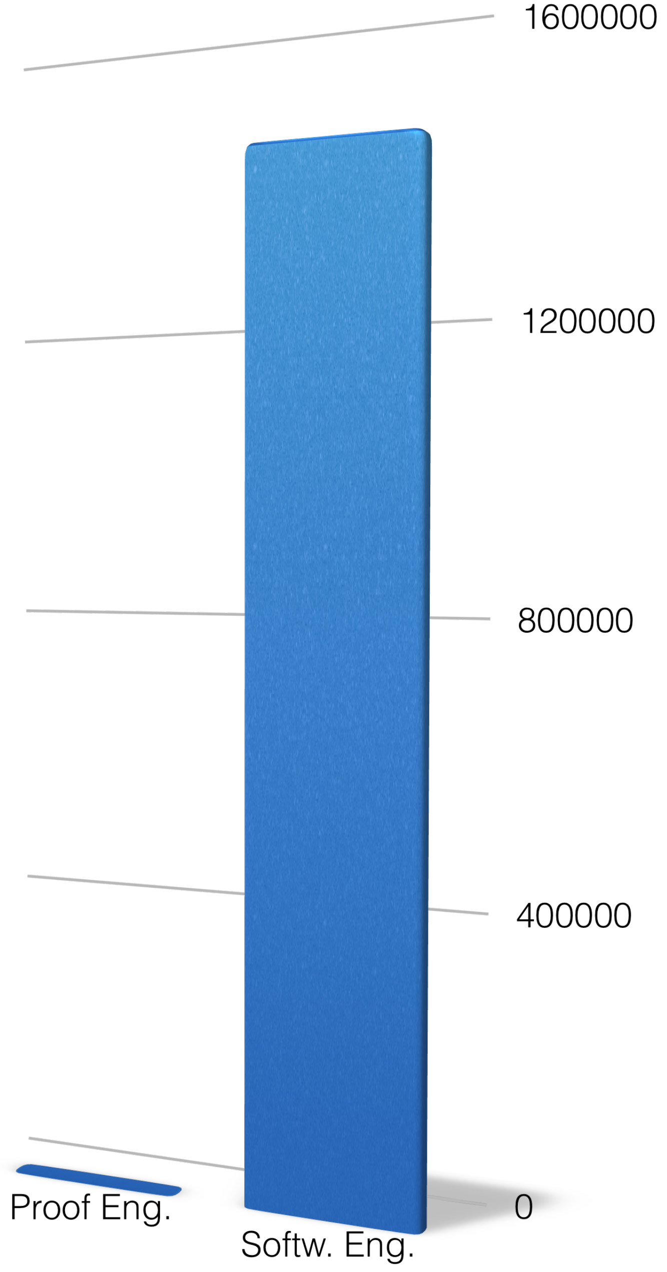
Proof Engineering is The Same

- Same kind of artefacts:
  - lemmas are functions, modules are modules
  - code gets big too
  - version control, regressions, refactoring and IDEs apply
- Same kind of problems
  - managing a large proof base over time
  - deliver a proof on time within budget
  - dependencies, interfaces, abstraction, etc



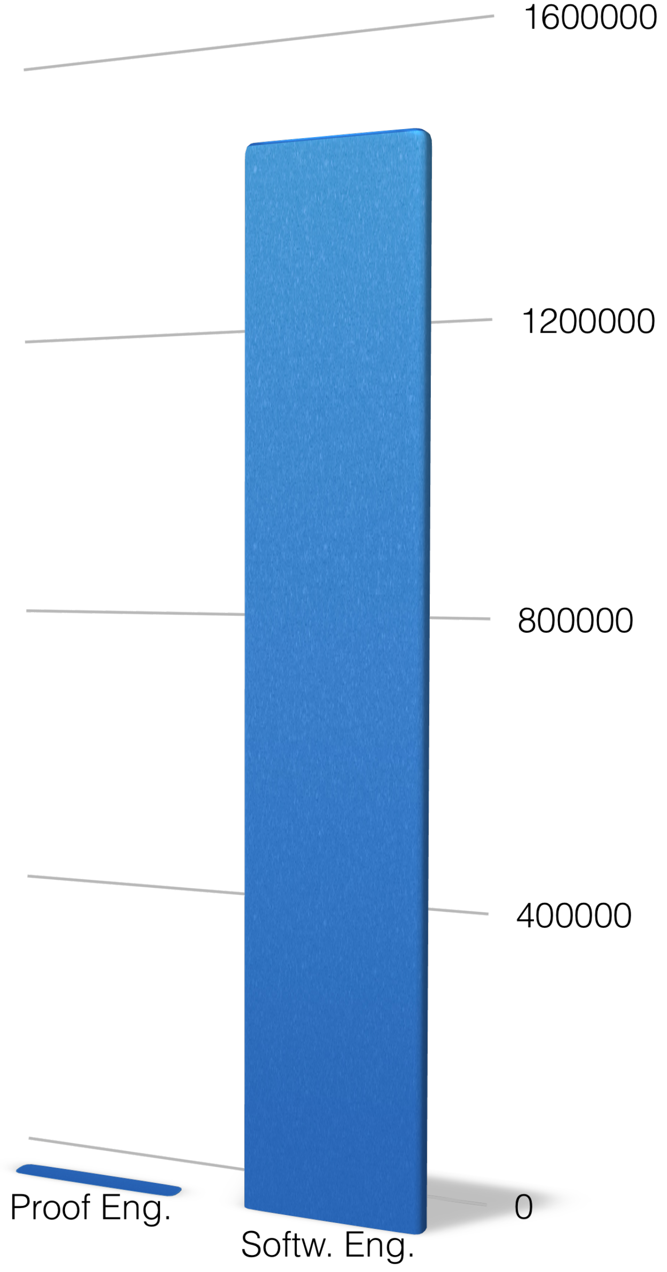
Proof Engineering is Different

- But: New Properties and Problems



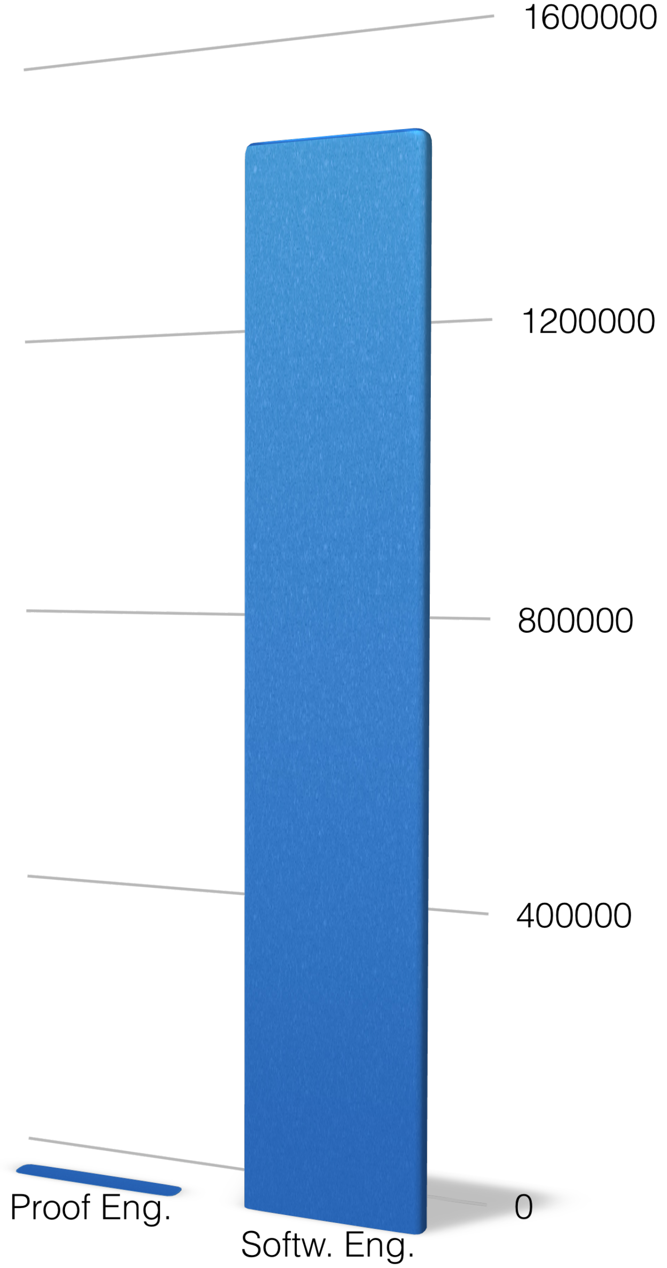
Proof Engineering is Different

- But: New Properties and Problems
  - Results are checkable
    - You know when you are done!
    - No testing
    - 95% proof: no such thing



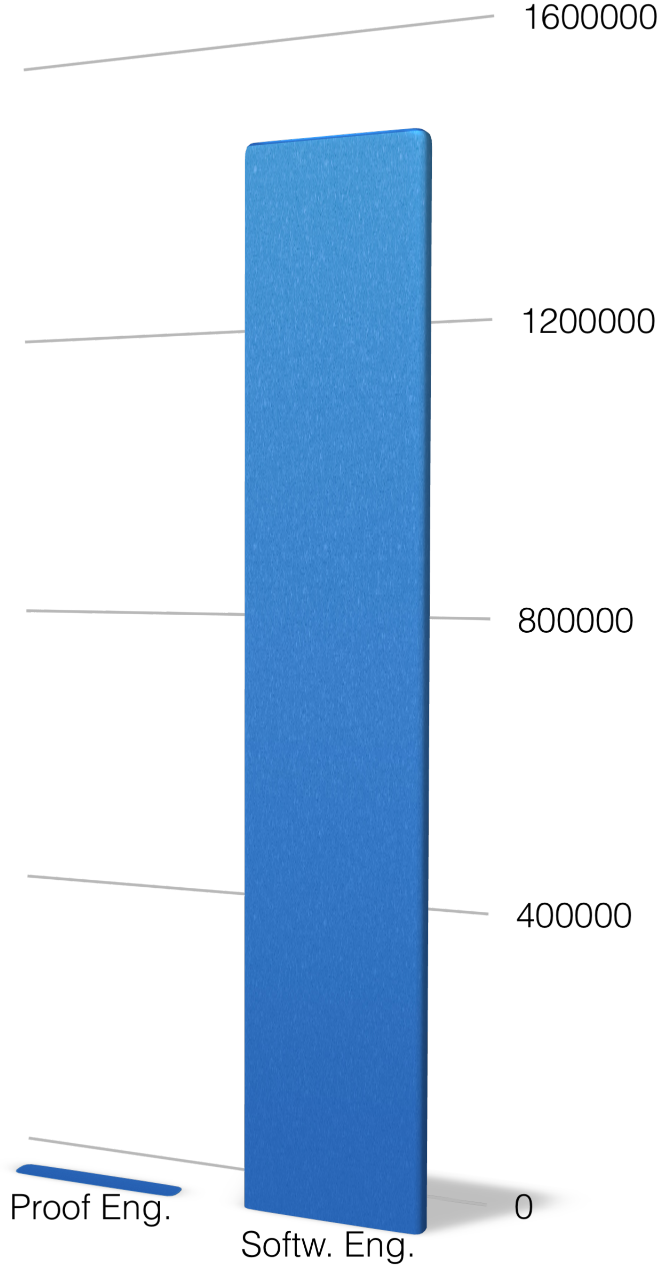
Proof Engineering is Different

- But: New Properties and Problems
  - Results are checkable
    - You know when you are done!
    - No testing
    - 95% proof: no such thing
  - More dead ends and iteration



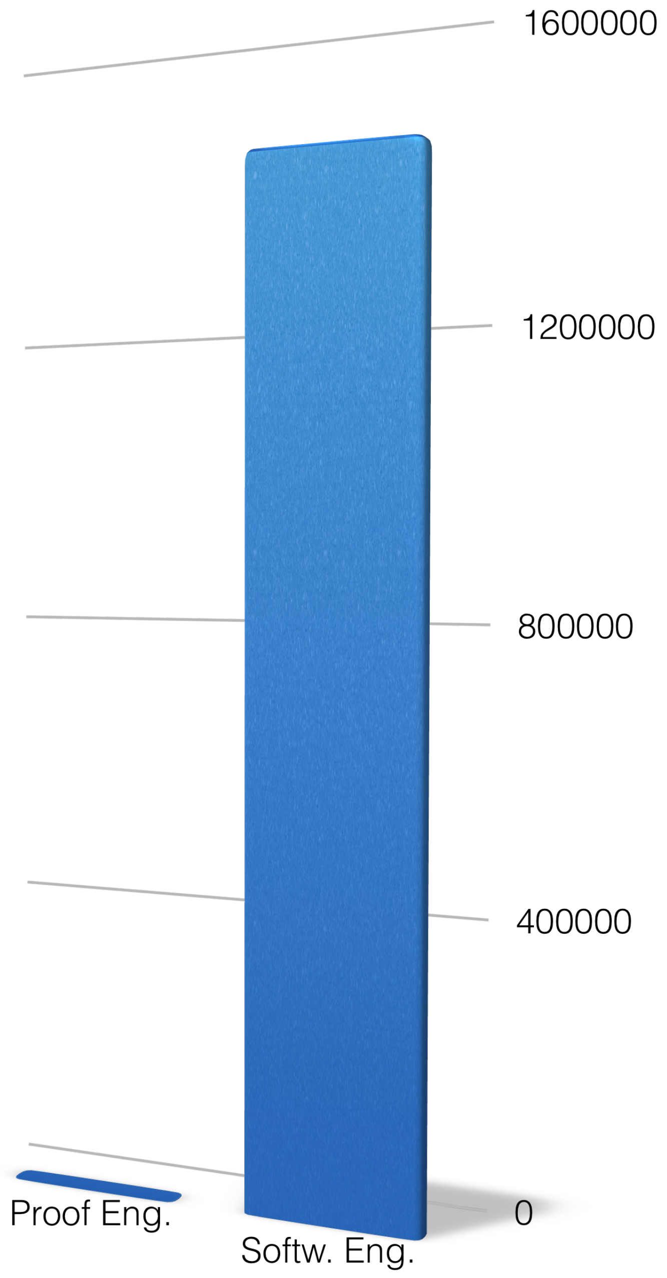
Proof Engineering is Different

- But: New Properties and Problems
  - Results are checkable
    - You know when you are done!
    - No testing
    - 95% proof: no such thing
  - More dead ends and iteration
  - 2nd order artefact
    - Performance less critical
    - Quality less critical
    - Proof Irrelevance



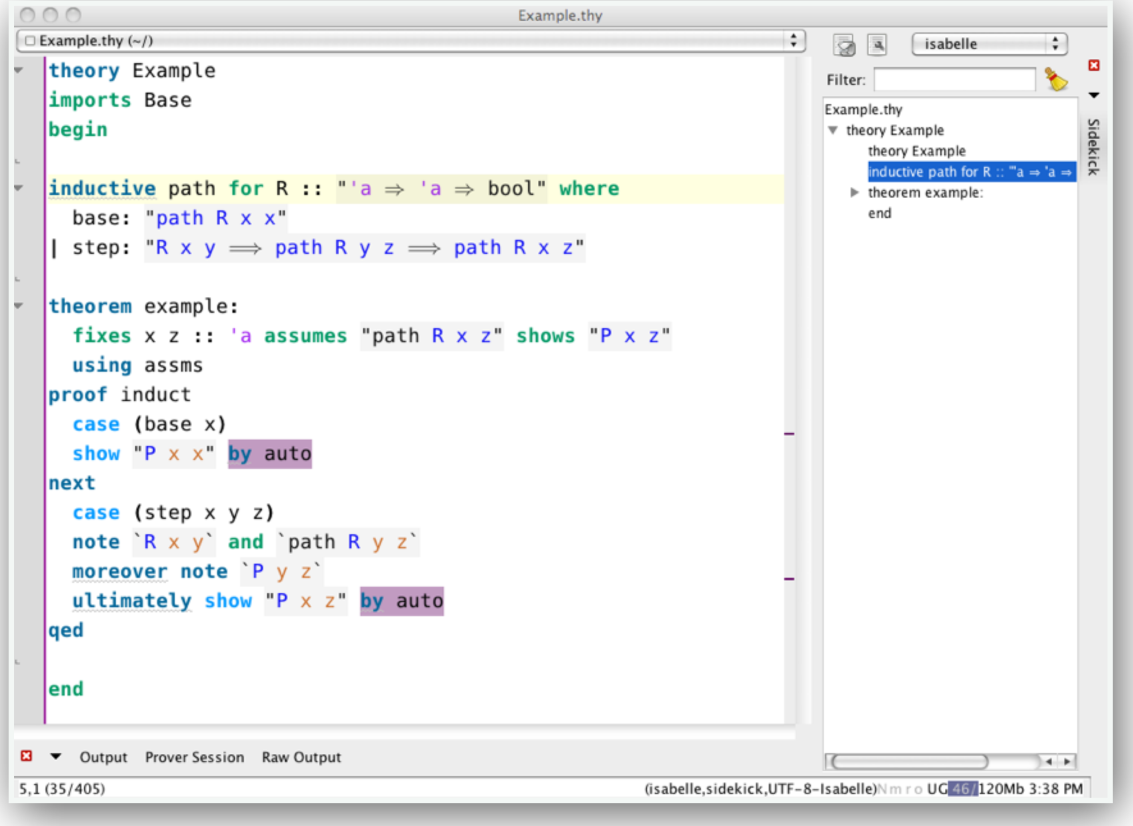
Proof Engineering is Different

- But: New Properties and Problems
  - Results are checkable
    - You know when you are done!
    - No testing
    - 95% proof: no such thing
  - More dead ends and iteration
  - 2nd order artefact
    - Performance less critical
    - Quality less critical
    - Proof Irrelevance
  - More semantic context
    - Much more scope for automation



Proof Engineering Tools

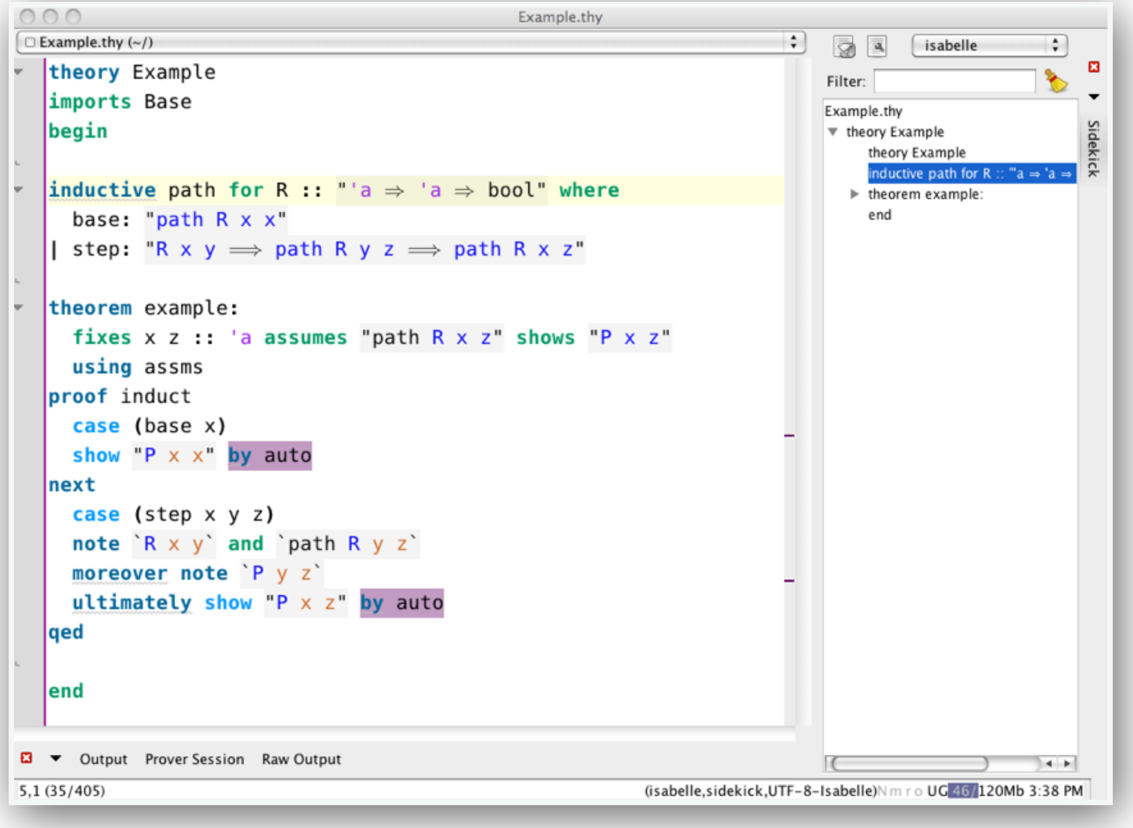
- User Interface
  - could proof IDEs be more powerful than code IDEs?
  - more semantic information
  - proof completion and suggestion?



```
theory Example
imports Base
begin

inductive path for R :: "'a ⇒ 'a ⇒ bool" where
  base: "path R x x"
| step: "R x y ⟹ path R y z ⟹ path R x z"

theorem example:
  fixes x z :: 'a assumes "path R x z" shows "P x z"
  using assms
proof induct
  case (base x)
  show "P x x" by auto
next
  case (step x y z)
  note `R x y` and `path R y z`
  moreover note `P y z`
  ultimately show "P x z" by auto
qed

end
```

Proof Engineering Tools

- User Interface
  - could proof IDEs be more powerful than code IDEs?
  - more semantic information
  - proof completion and suggestion?



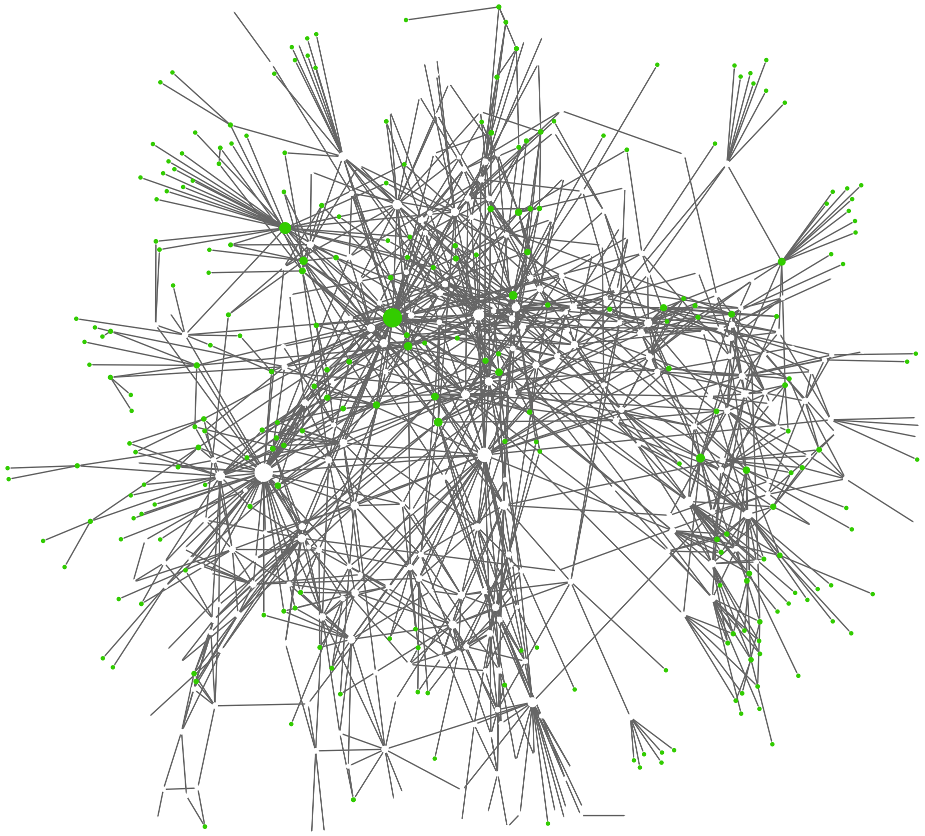
```
theory Example
imports Base
begin

inductive path for R :: "'a ⇒ 'a ⇒ bool" where
  base: "path R x x"
| step: "R x y ⟹ path R y z ⟹ path R x z"

theorem example:
  fixes x z :: 'a assumes "path R x z" shows "P x z"
  using assms
proof induct
  case (base x)
  show "P x x" by auto
next
  case (step x y z)
  note `R x y` and `path R y z`
  moreover note `P y z`
  ultimately show "P x z" by auto
qed

end
```

```
5,1 (35/405)                    (isabelle,sidekick,UTF-8-Isabelle)N m r o UG 46/120Mb 3:38 PM
```

- Refactoring
  - less constrained,
    new kinds of refactoring possible, e.g.
    - move to best position in library
    - generalise lemma
    - recognise proof patterns

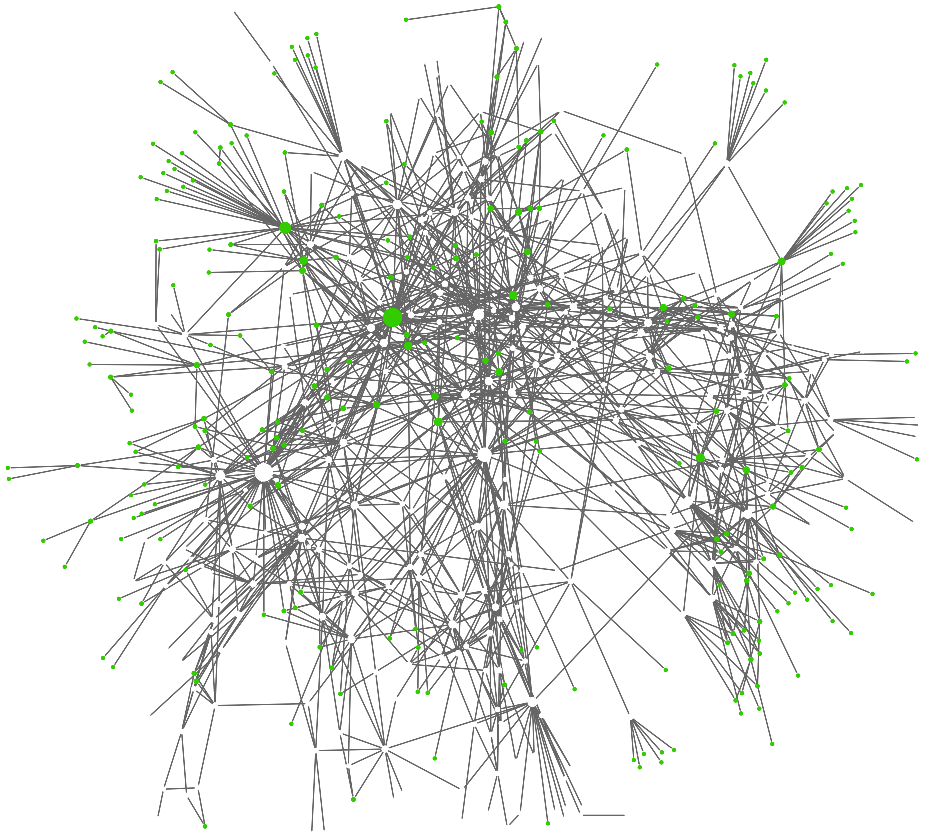
Proof Patterns

- Large-scale Libraries
  - architecture:
    - layers, modules, components, abstractions, genericity
  - proof interfaces
  - proof patterns



Proof Patterns

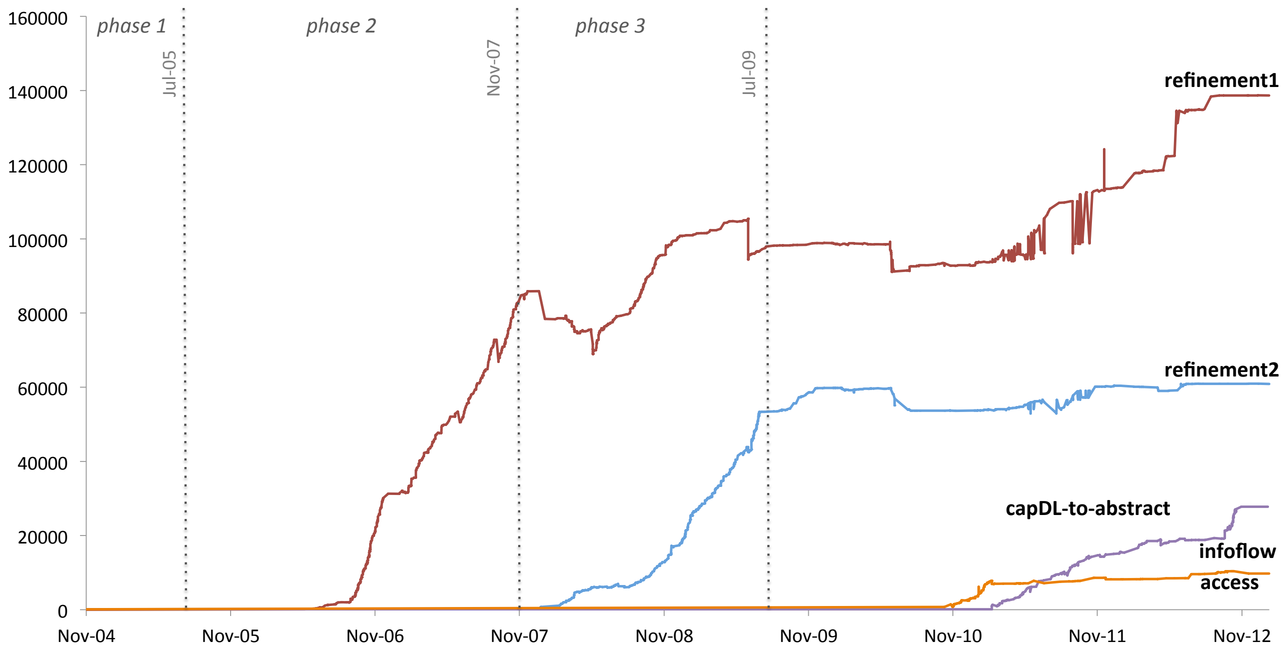
- Large-scale Libraries
  - architecture:
    - layers, modules, components, abstractions, genericity
  - proof interfaces
  - proof patterns



- Technical Debt
  - what does a clean, maintainable proof look like?
  - which techniques will make future change easier?
  - readability important? is documentation?

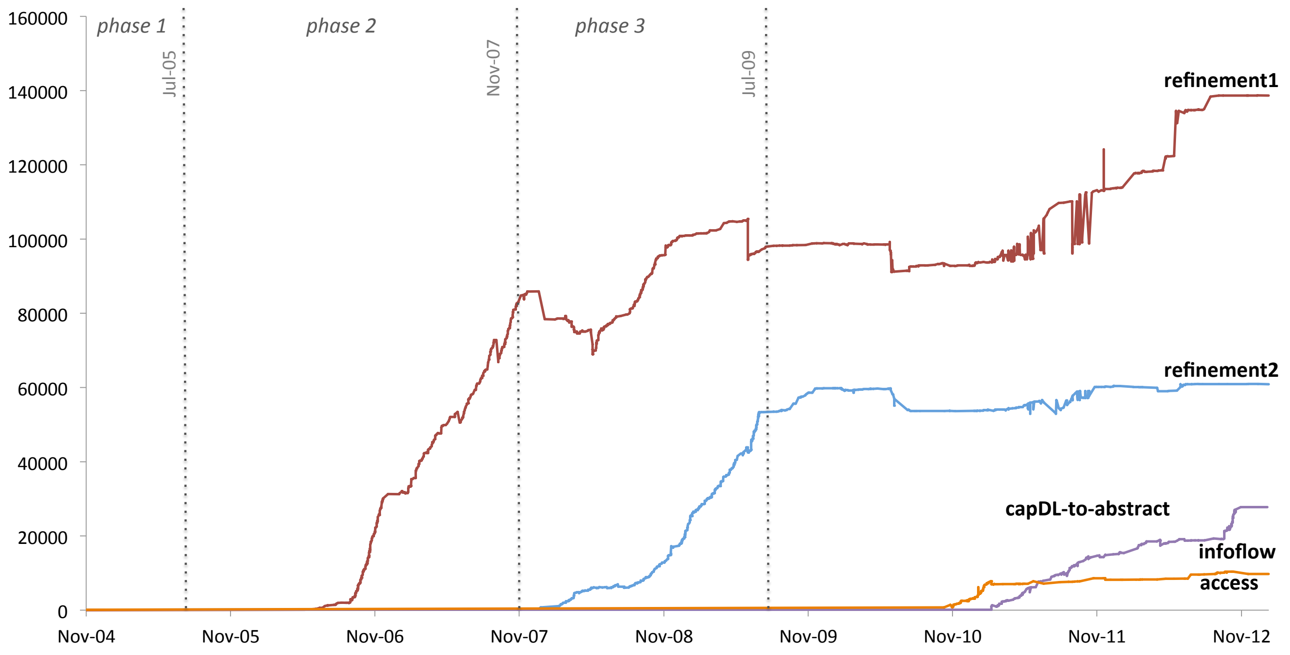
Proof Engineering "Laws"

- Are there Proof Engineering Laws?



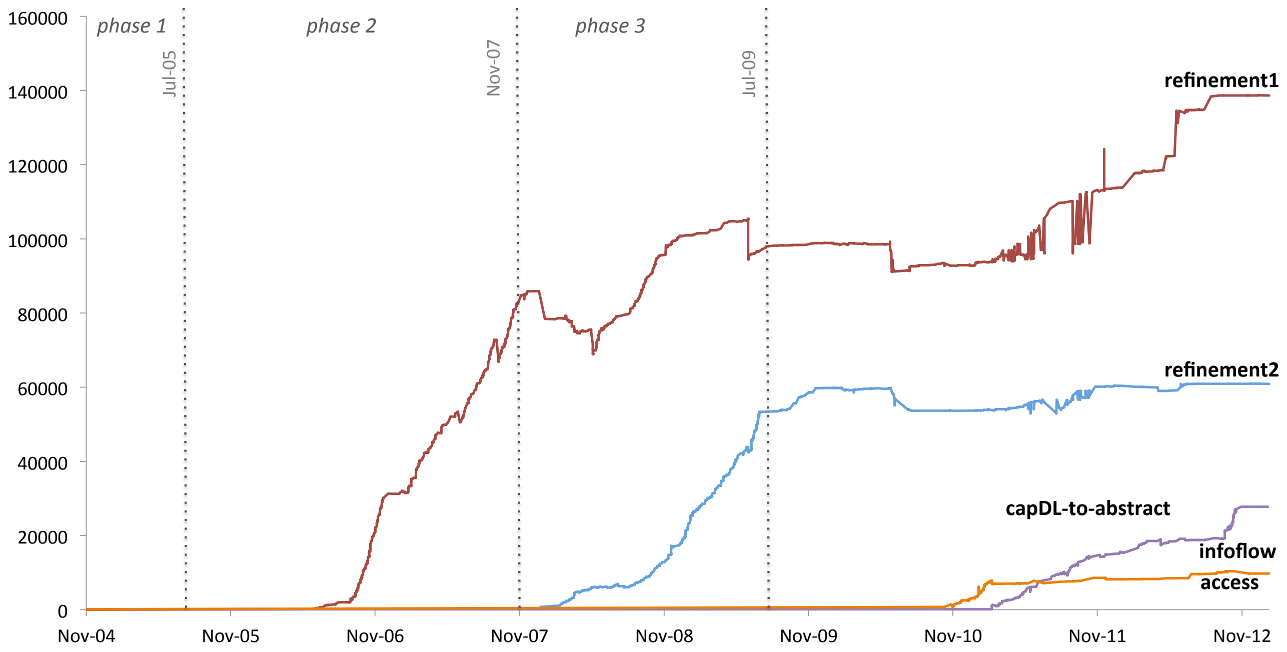
Proof Engineering "Laws"

- Are there Proof Engineering Laws?
  - Proofs always become larger and more complex over time. (from Cope's rule)



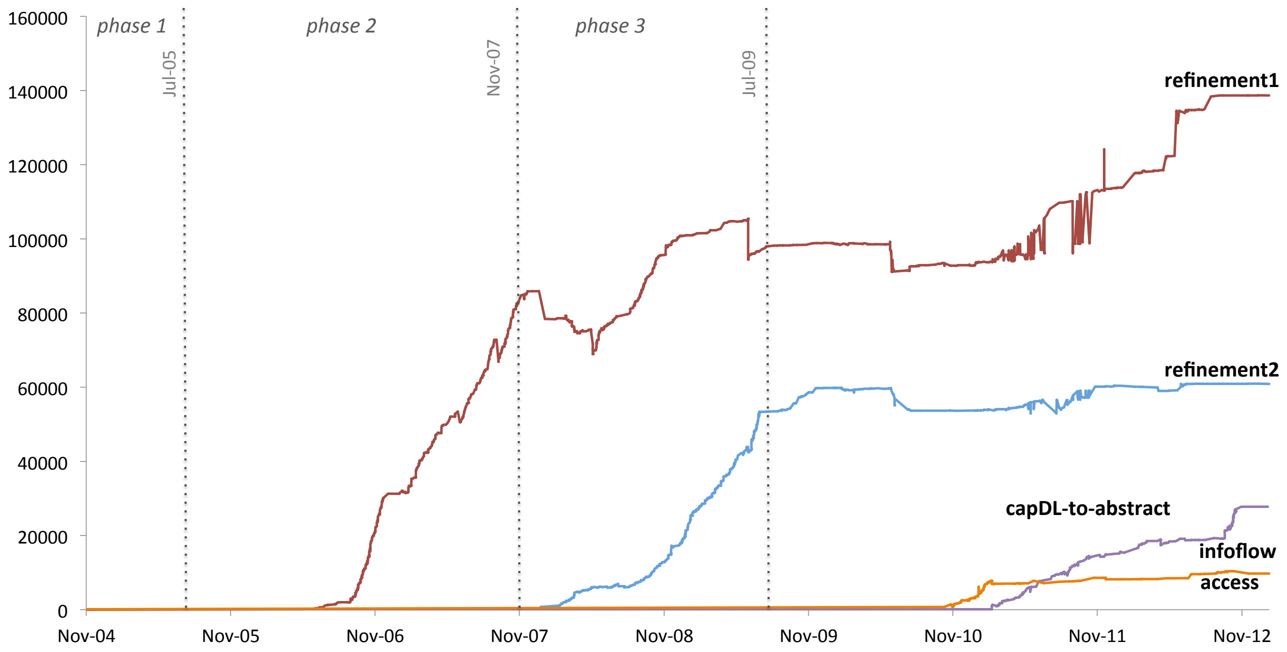
Proof Engineering "Laws"

- Are there Proof Engineering Laws?
  - Proofs always become larger and more complex over time. (from Cope's rule)
  - Adding manpower to a late proof project makes it later. (from Brooks' law)



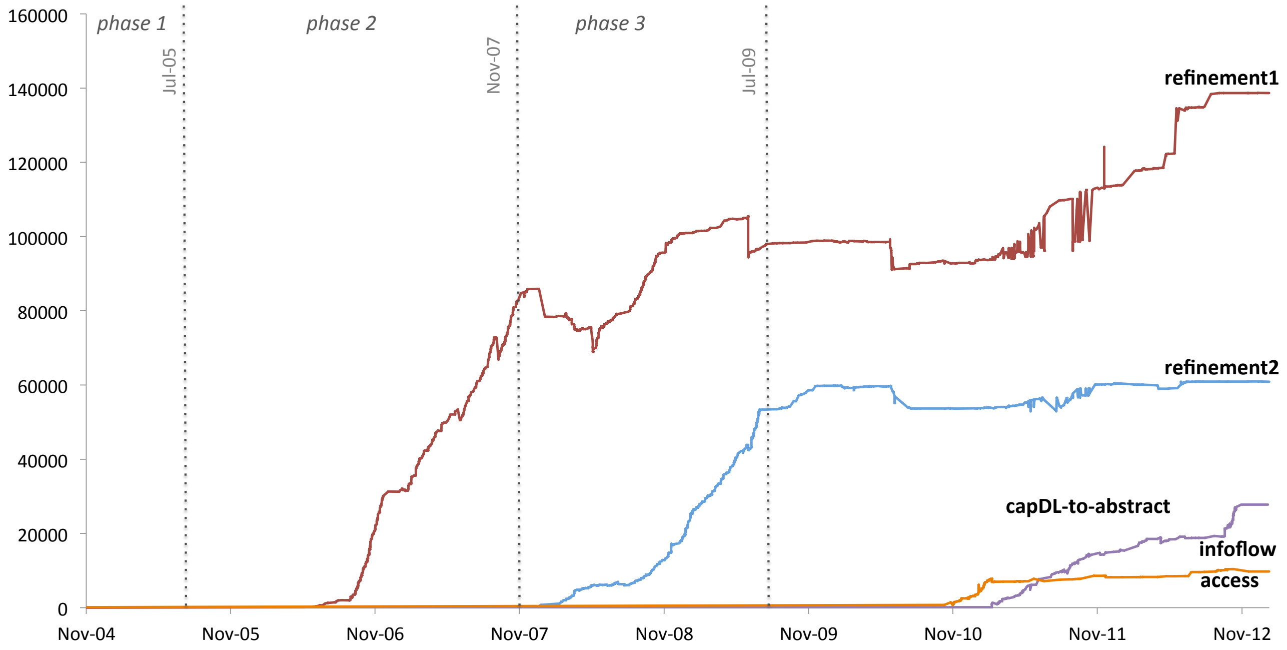
Proof Engineering "Laws"

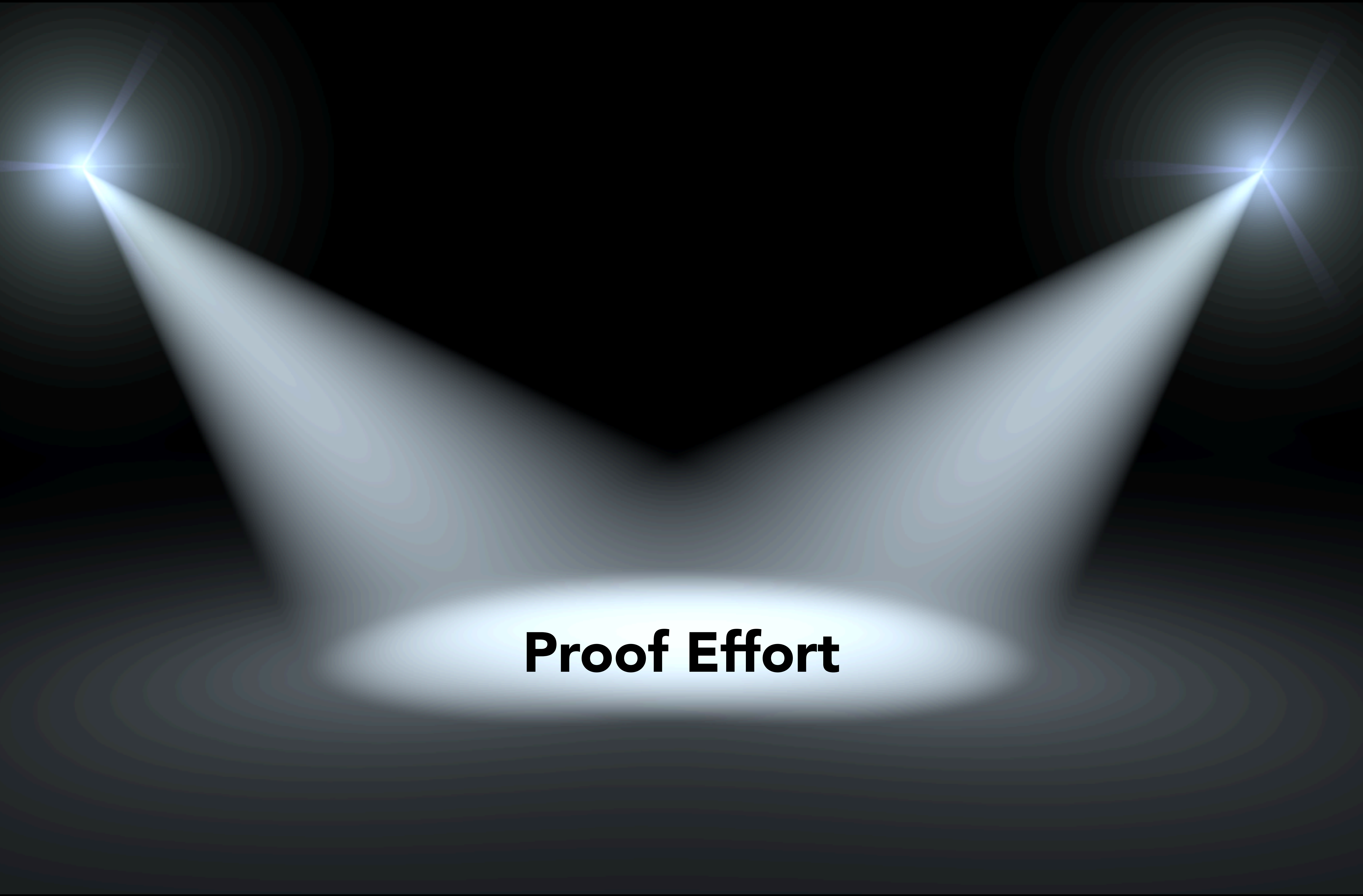
- Are there Proof Engineering Laws?
  - Proofs always become larger and more complex over time. (from Cope's rule)
  - Adding manpower to a late proof project makes it later. (from Brooks' law)
  - You cannot reduce the complexity of a given proof beyond a certain point. Once you've reached that point, you can only shift the burden around. (from Tesler's law)



Proof Engineering "Laws"

- Are there Proof Engineering Laws?
  - Proofs always become larger and more complex over time. (from Cope's rule)
  - Adding manpower to a late proof project makes it later. (from Brooks' law)
  - You cannot reduce the complexity of a given proof beyond a certain point. Once you've reached that point, you can only shift the burden around. (from Tesler's law)
  - Are they true?





Proof Effort

Predictions

Can we predict for proofs:

- how large will it be?
- how long will it take?
- how much will it cost?

Predictions

Can we predict for proofs:

- how large will it be?
- how long will it take?

Of course not.

Many hard problems look deceptively easy.

Predictions

Can we predict for proofs:

- how large will it be?
- how long will it take?

Of course not.

Many hard problems look deceptively easy.

But maybe for program verification?

At least statistically, some of the time?

Predictions

Can we predict for proofs:

- how large will it be?
- how long will it take?

Of course

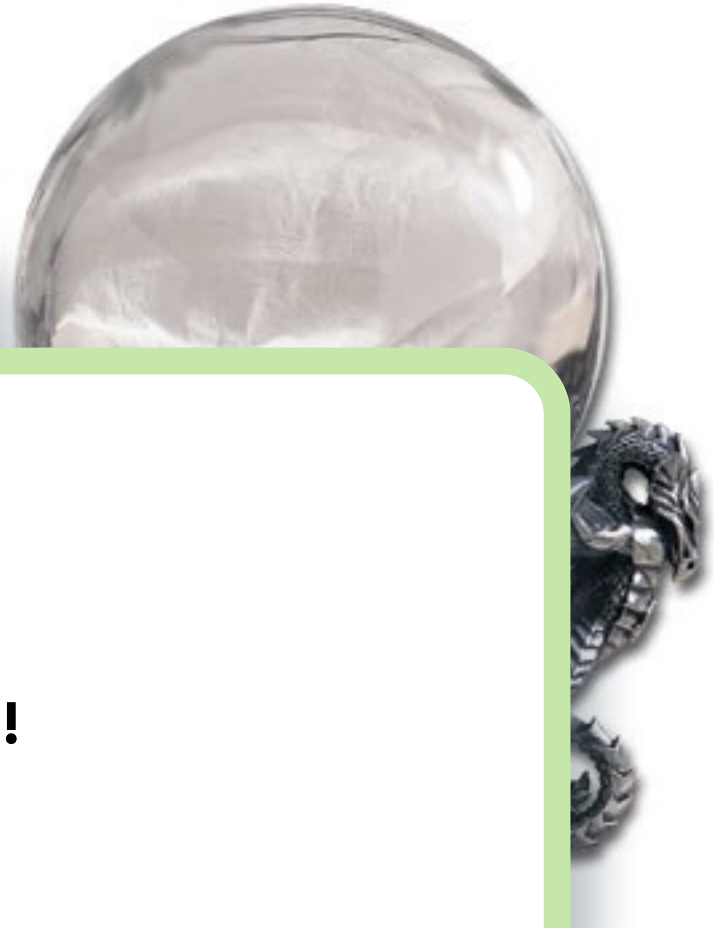
Many har

But ma

At least

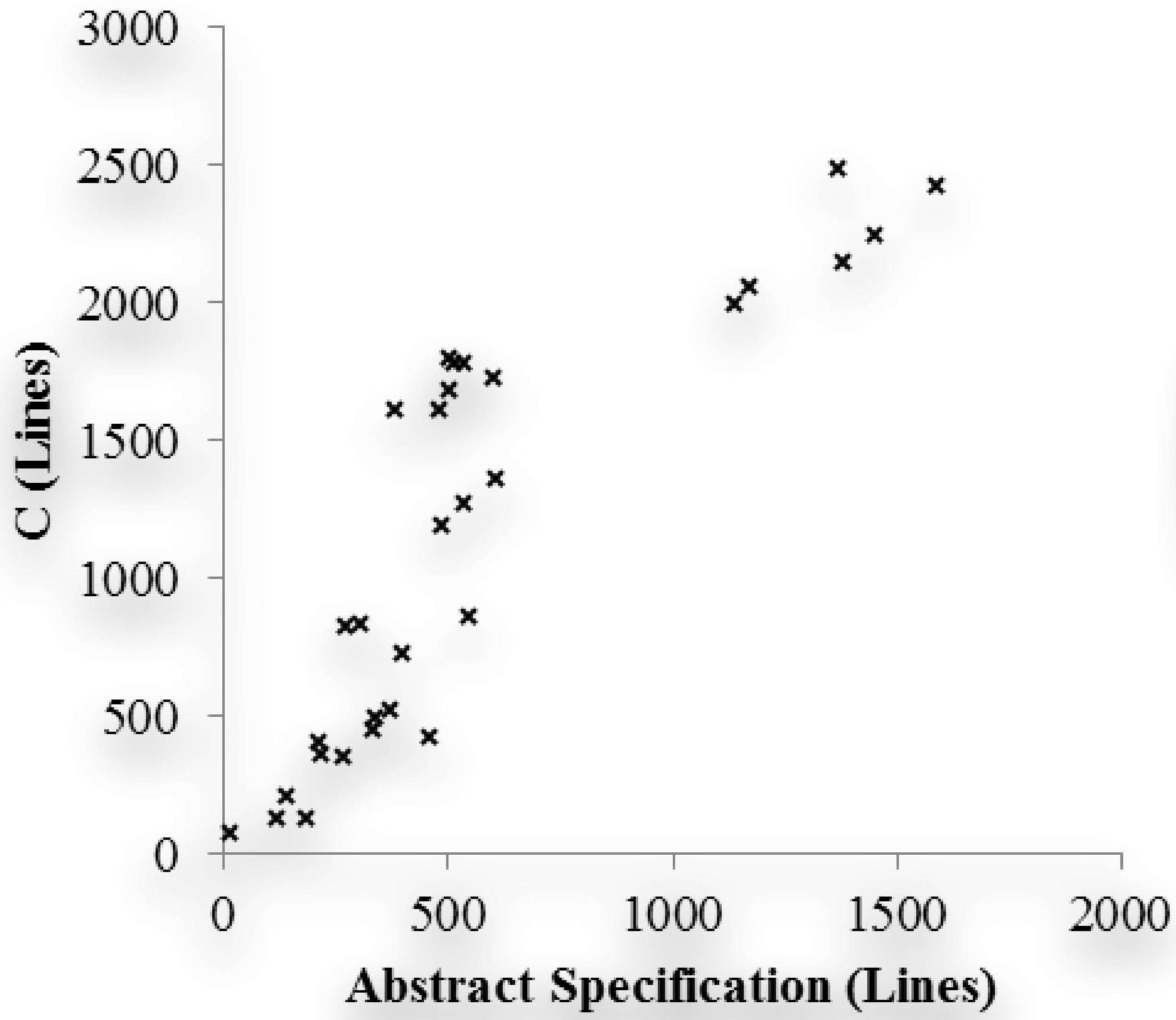
We have large proofs.

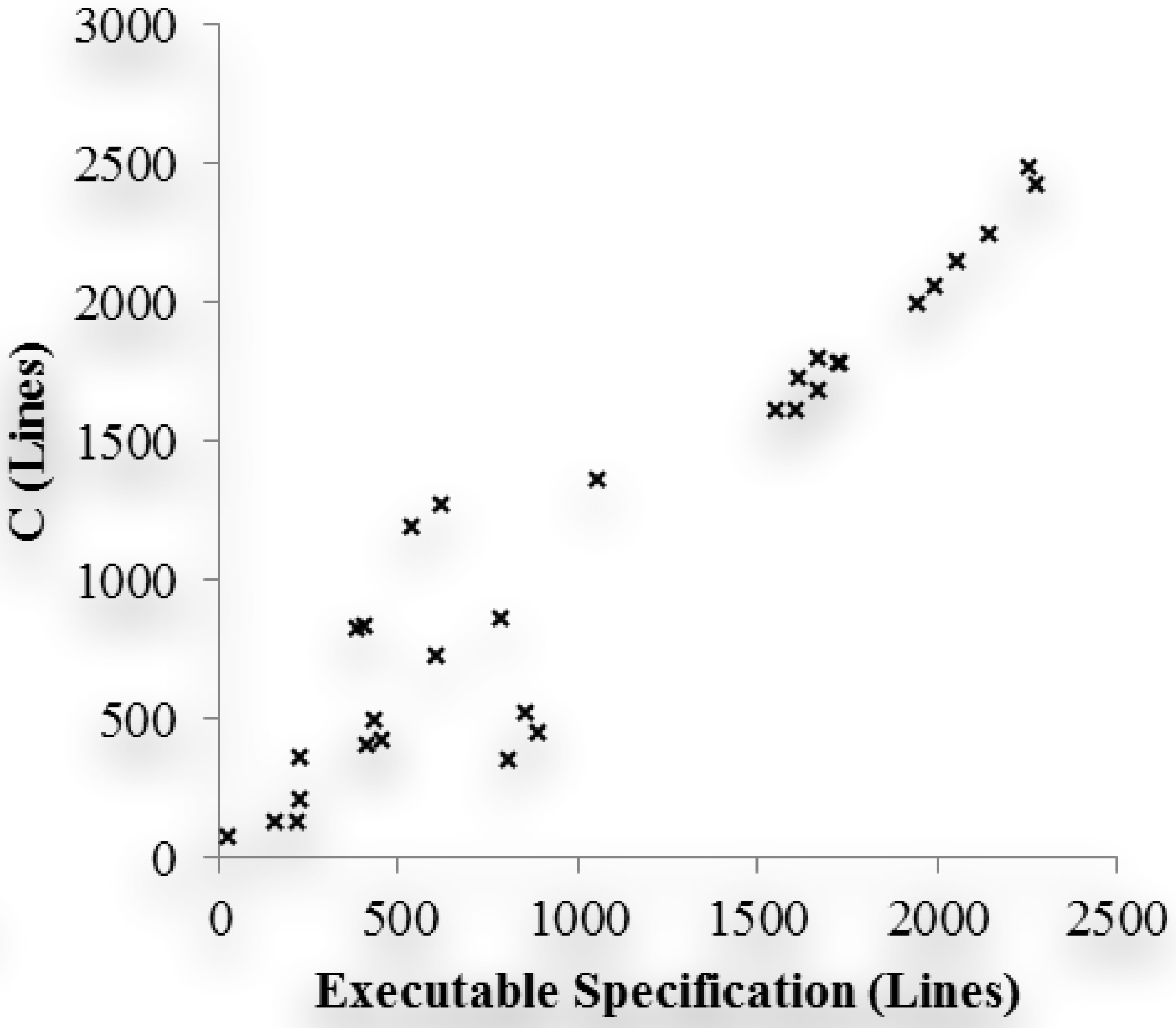
Let’s crunch some data!



Spec and Code Size

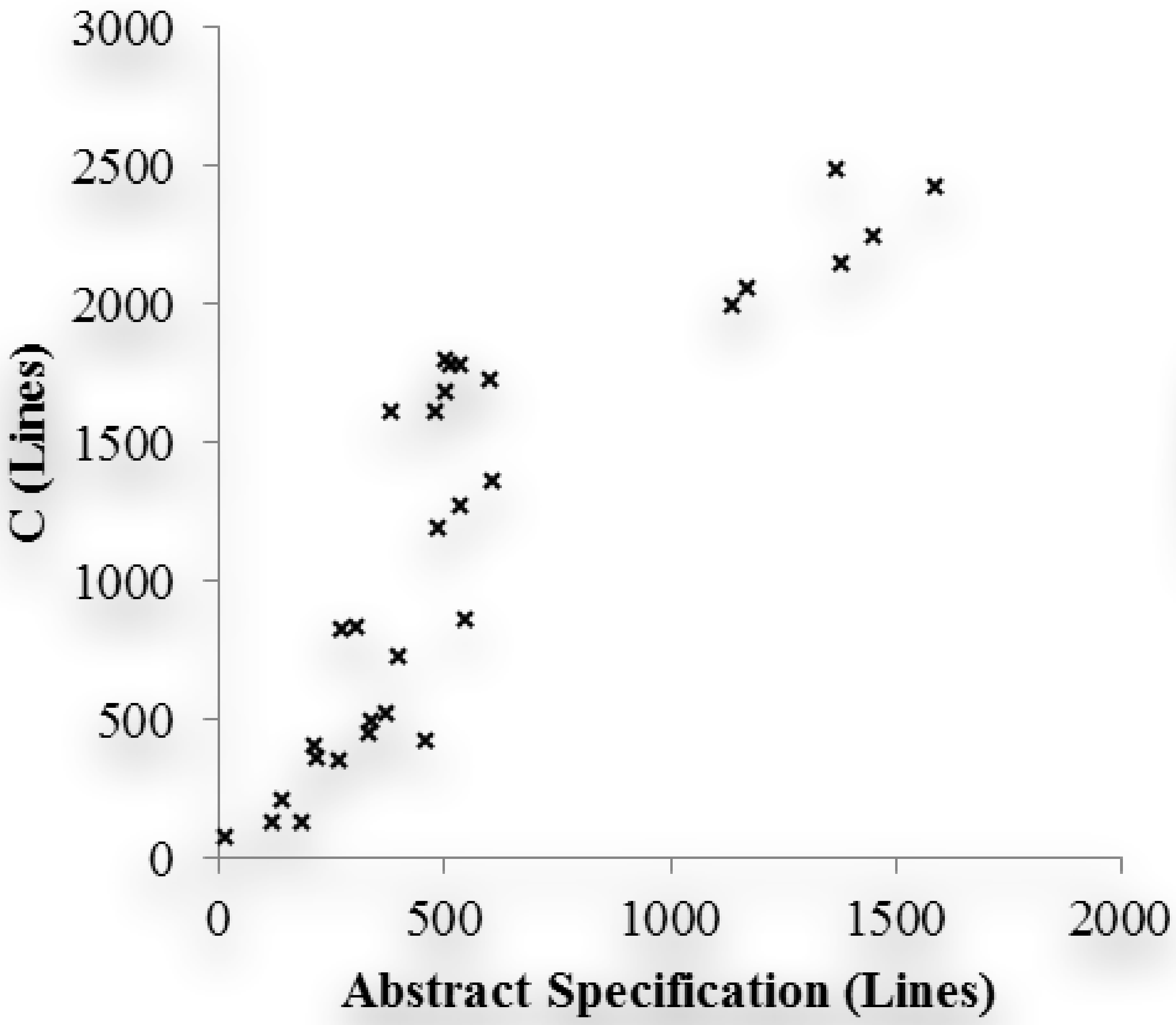
Specification Size and Code Size are extremely well correlated in seL4.

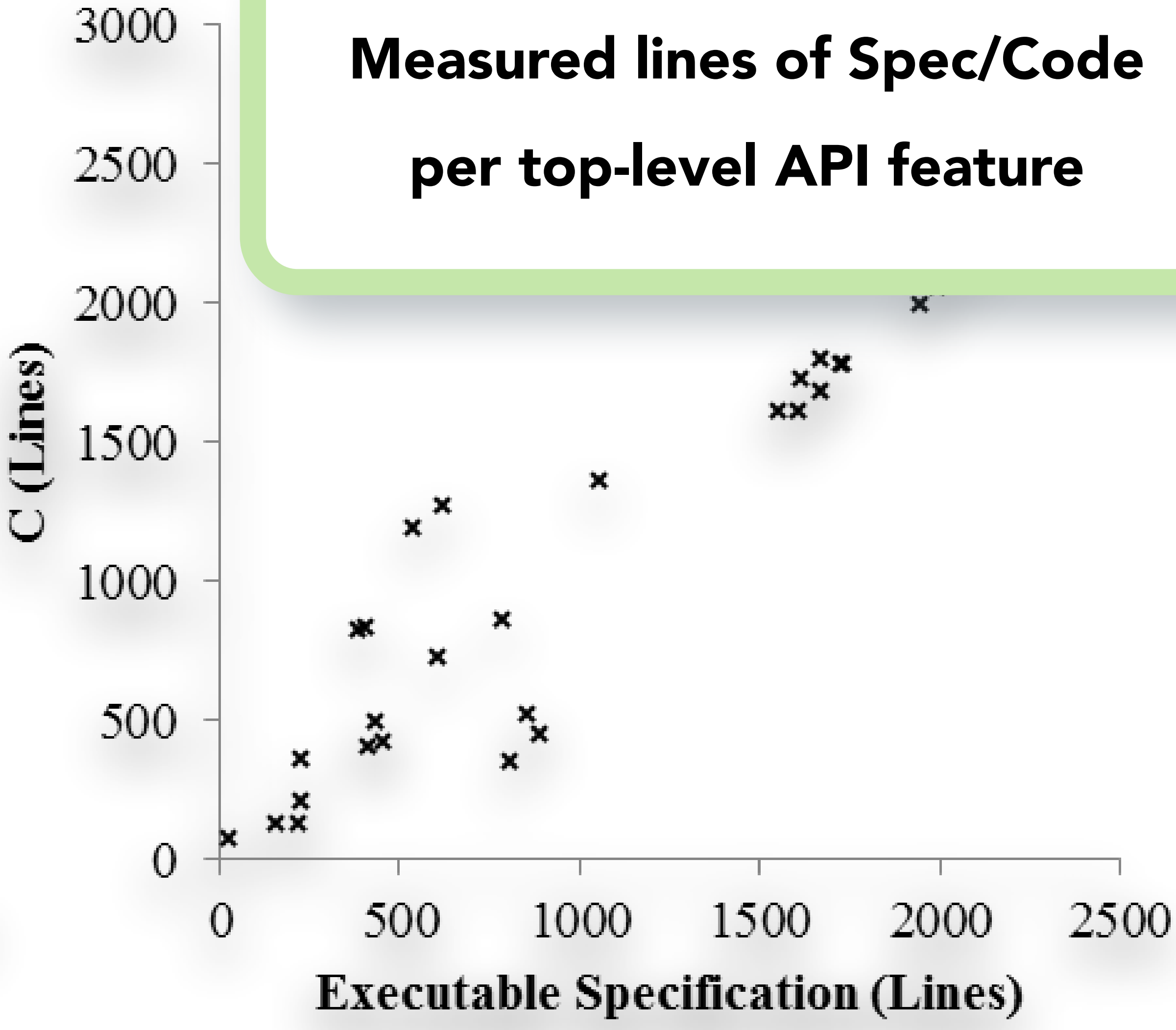




Spec and Code Size

Specification Size and Code Size are extremely well correlated in seL4.





Measured lines of Spec/Code

per top-level API feature

Effort vs Proof Size

Proof Effort = work time spent on a proof ≈ money

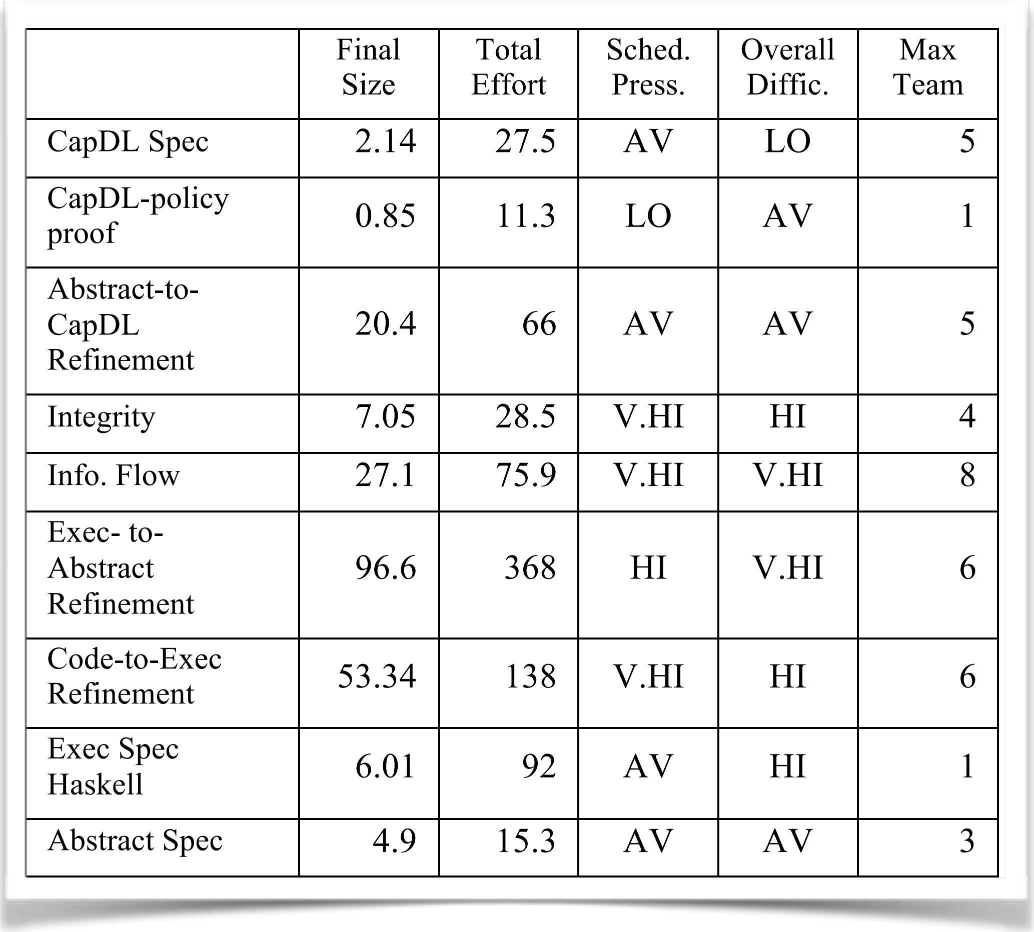
Is Proof Effort related to Proof Size?

Are there small proofs that take very long?

Large proofs that were quick to write?

Manually reconstructed effort from repo logs, meeting notes, and progress reports.

Measured proof size.



	Final Size	Total Effort	Sched. Press.	Overall Diffic.	Max Team
CapDL Spec	2.14	27.5	AV	LO	5
CapDL-policy proof	0.85	11.3	LO	AV	1
Abstract-to-CapDL Refinement	20.4	66	AV	AV	5
Integrity	7.05	28.5	V.HI	HI	4
Info. Flow	27.1	75.9	V.HI	V.HI	8
Exec- to-Abstract Refinement	96.6	368	HI	V.HI	6
Code-to-Exec Refinement	53.34	138	V.HI	HI	6
Exec Spec Haskell	6.01	92	AV	HI	1
Abstract Spec	4.9	15.3	AV	AV	3

Effort vs Proof Size

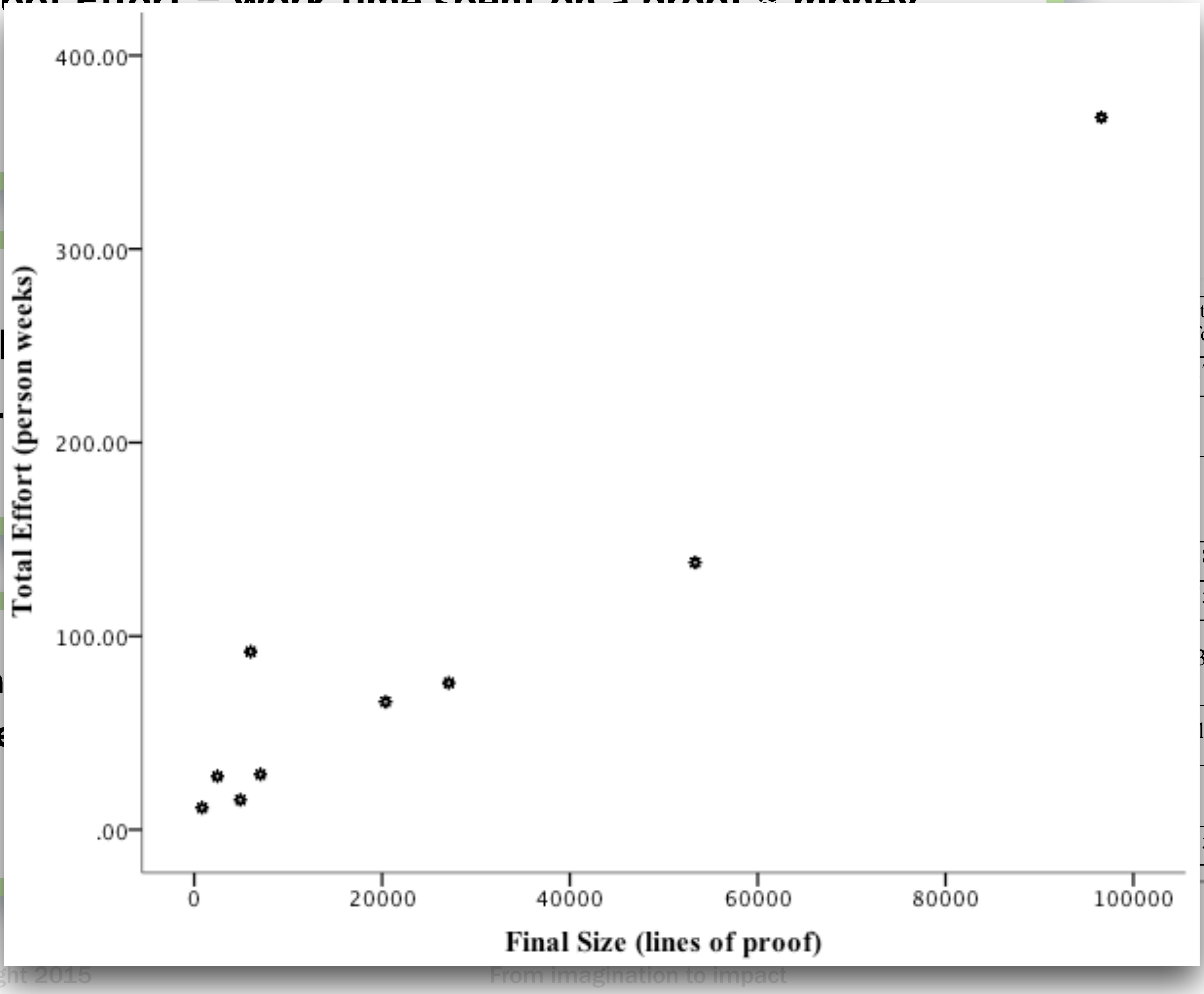
Proof Effort = work time spent on a proof ≈ money

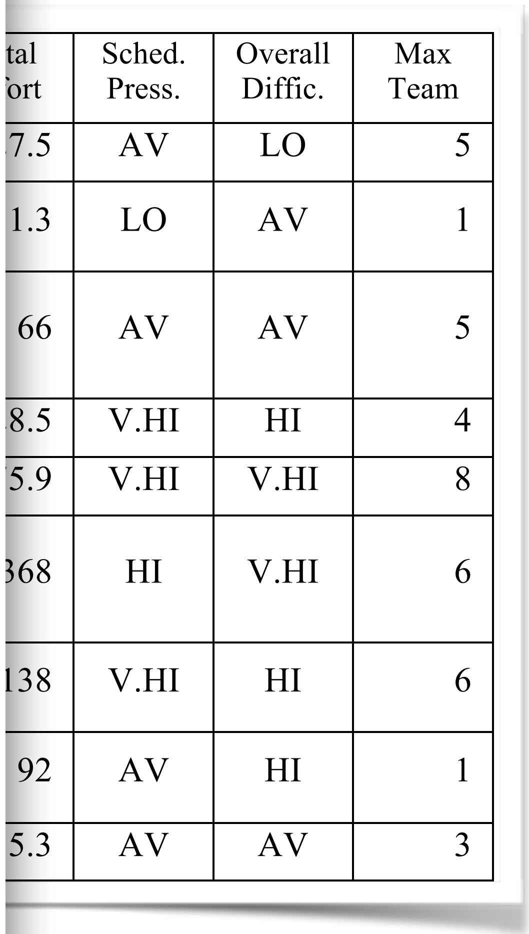
Are t

Lar

Manua

me





tal fort	Sched. Press.	Overall Diffic.	Max Team
7.5	AV	LO	5
1.3	LO	AV	1
66	AV	AV	5
8.5	V.HI	HI	4
5.9	V.HI	V.HI	8
368	HI	V.HI	6
138	V.HI	HI	6
92	AV	HI	1
5.3	AV	AV	3

Effort vs Proof Size

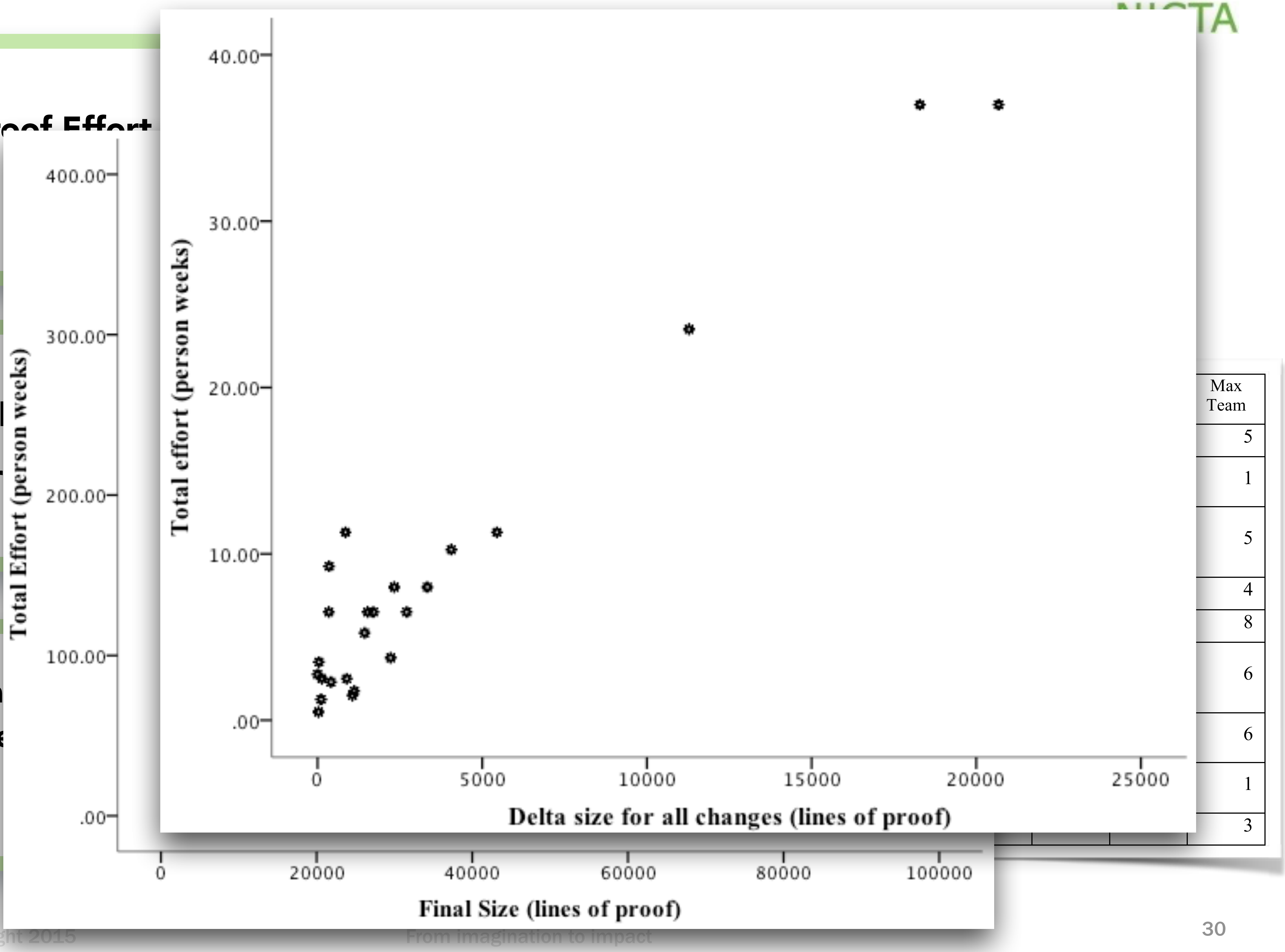
Proof Effort

Are t

Lar

Manua

me



Max Team
5
1
5
4
8
6
6
1
3

Effort vs Proof Size

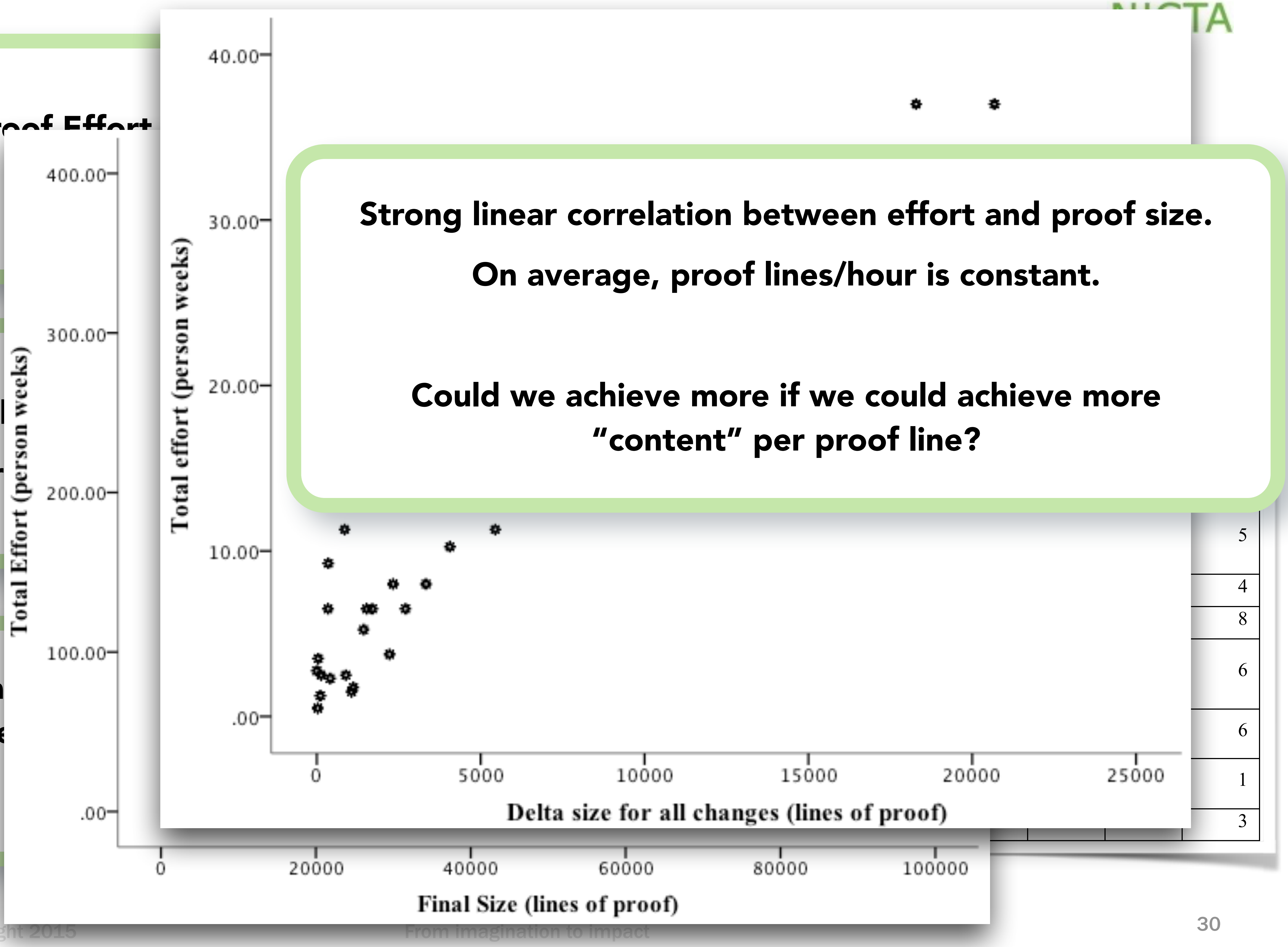
Proof Effort

Are t

Lar

Manua

me



Strong linear correlation between effort and proof size.

On average, proof lines/hour is constant.

Could we achieve more if we could achieve more "content" per proof line?

Spec Size and Proof Size

Spec Size and Proof Size

If proof size = effort/cost,

is there a leading indicator for proof size?

Spec Size and Proof Size

If proof size = effort/cost,

is there a leading indicator for proof size?

How about specification/lemma statement size or complexity?

Spec Size and Proof Size

If proof size = effort/cost,

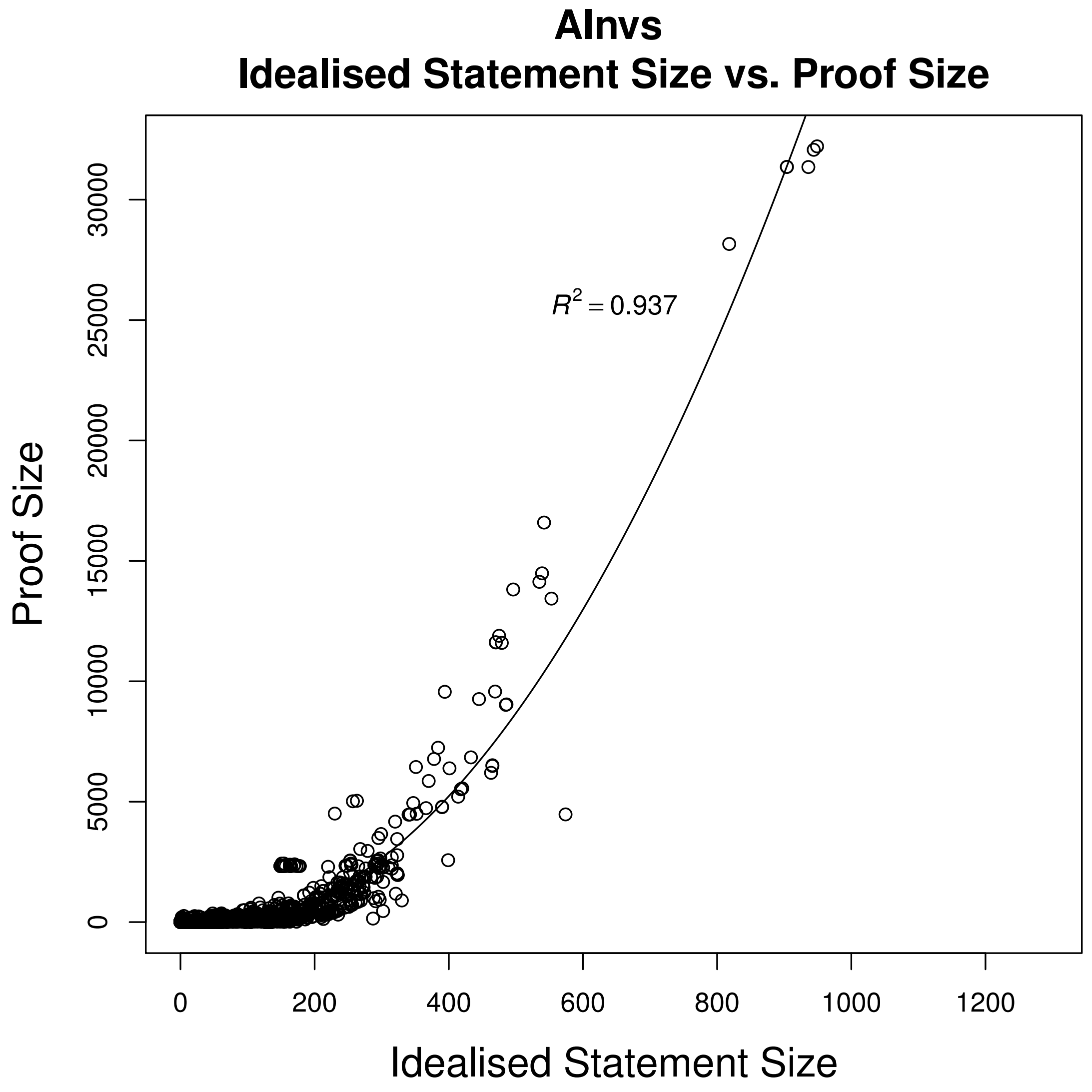
is there a leading indicator for proof size?

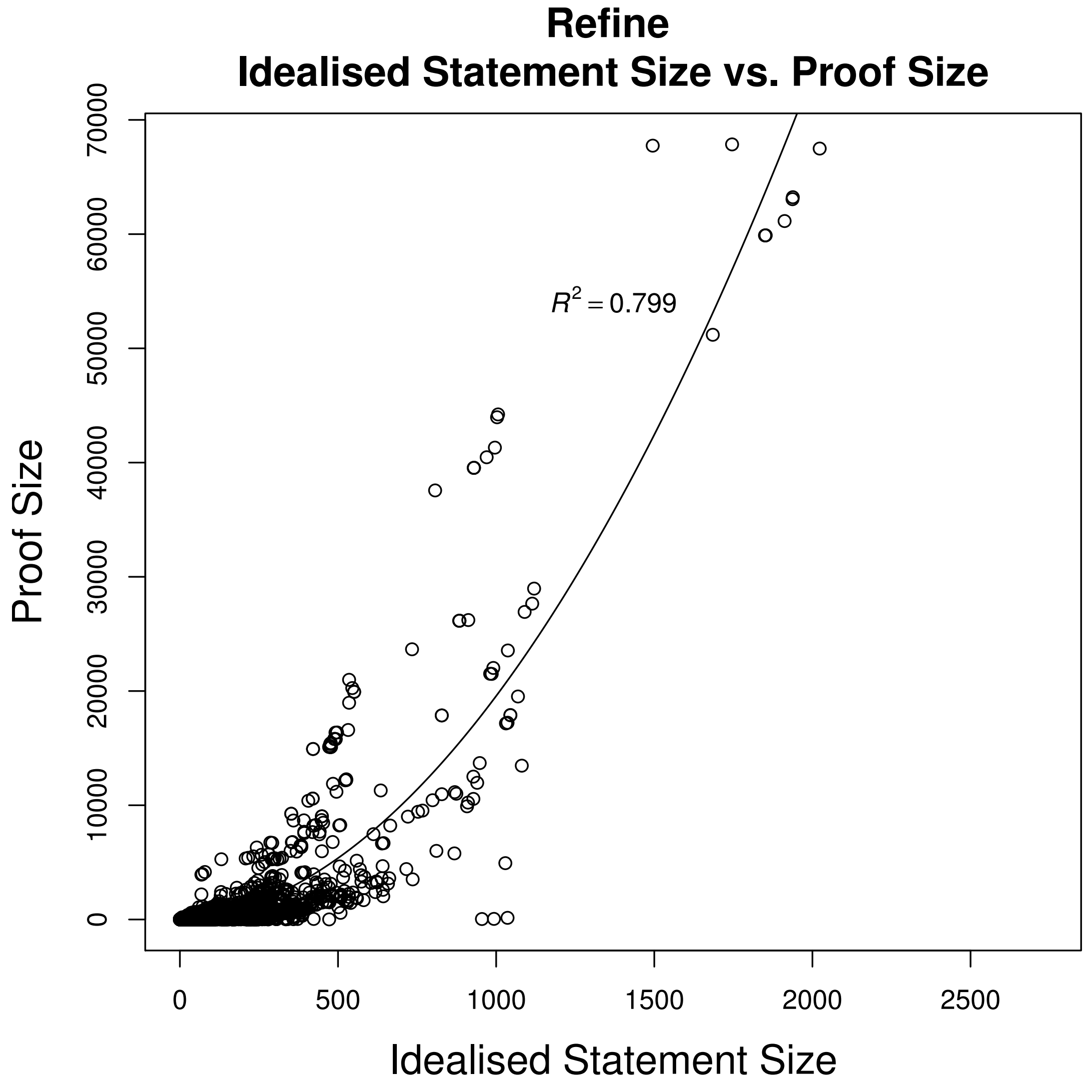
How about specification/lemma statement size or complexity?

Measured: lemma statement size by number of constants, recursively.

Measured: lemma proof script size, recursively for used lemmas.

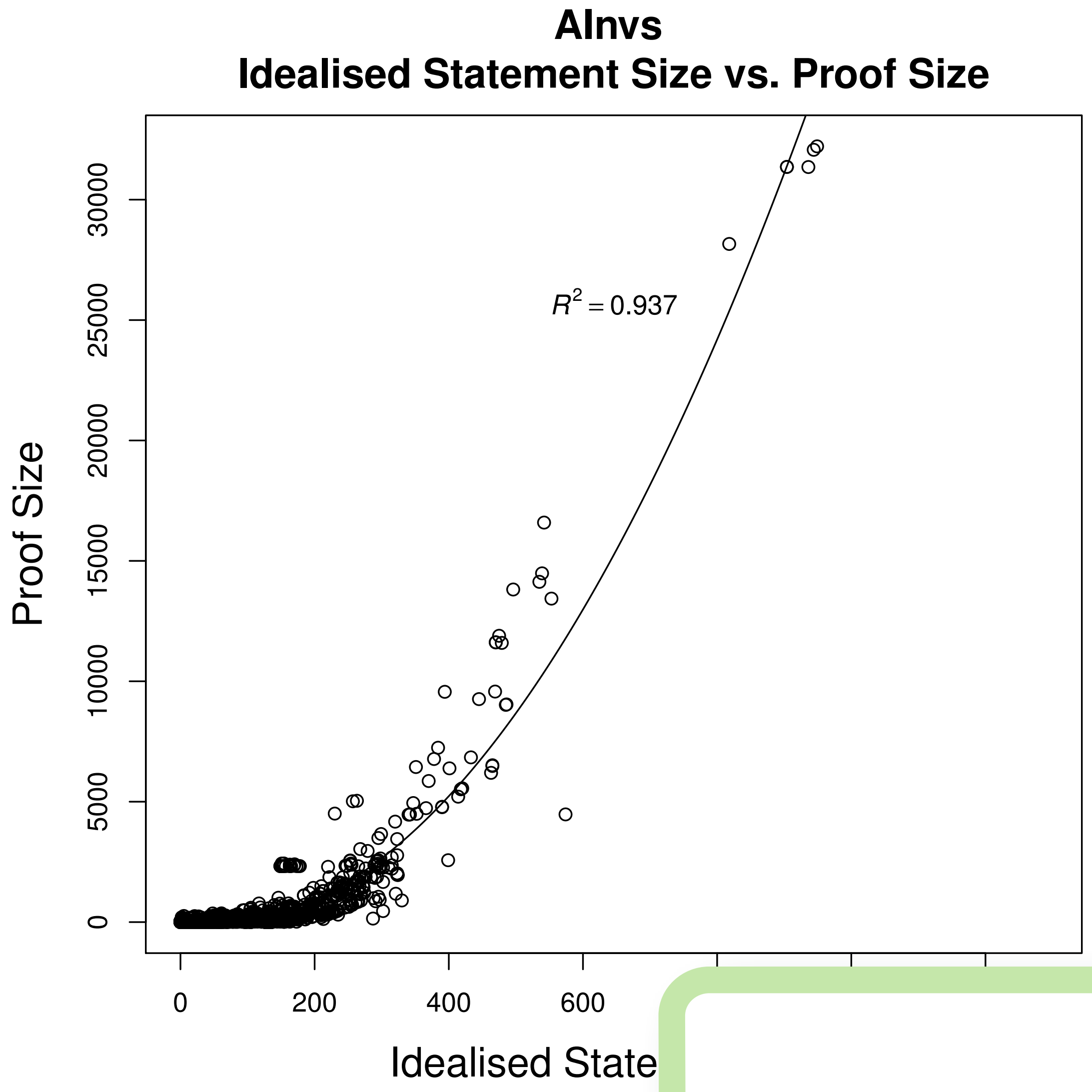
Spec Size vs Proof Size



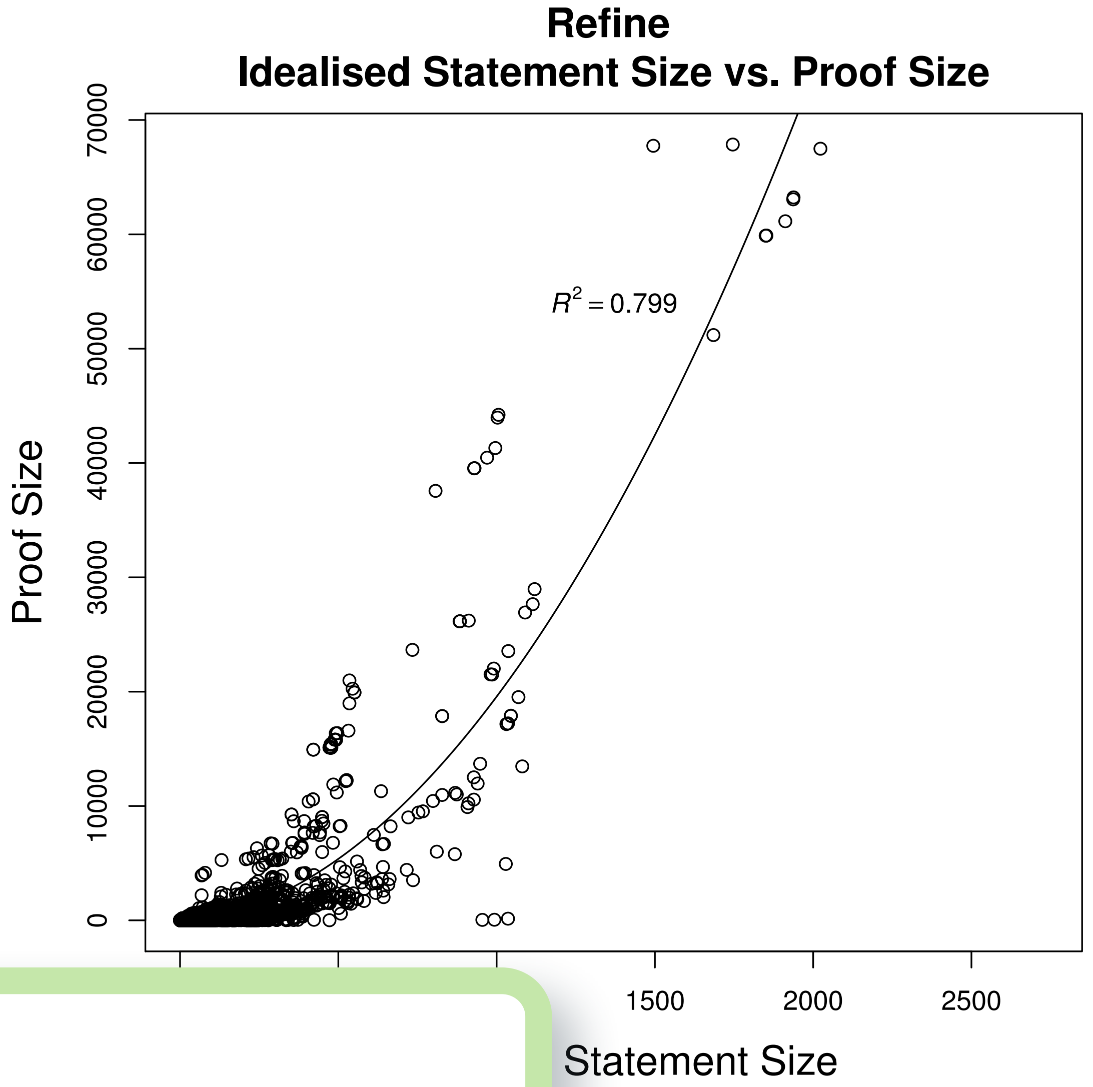


Idealised Statement Size: do not count unused constants

Spec Size vs Proof Size



AInvs
Idealised Statement Size vs. Proof Size

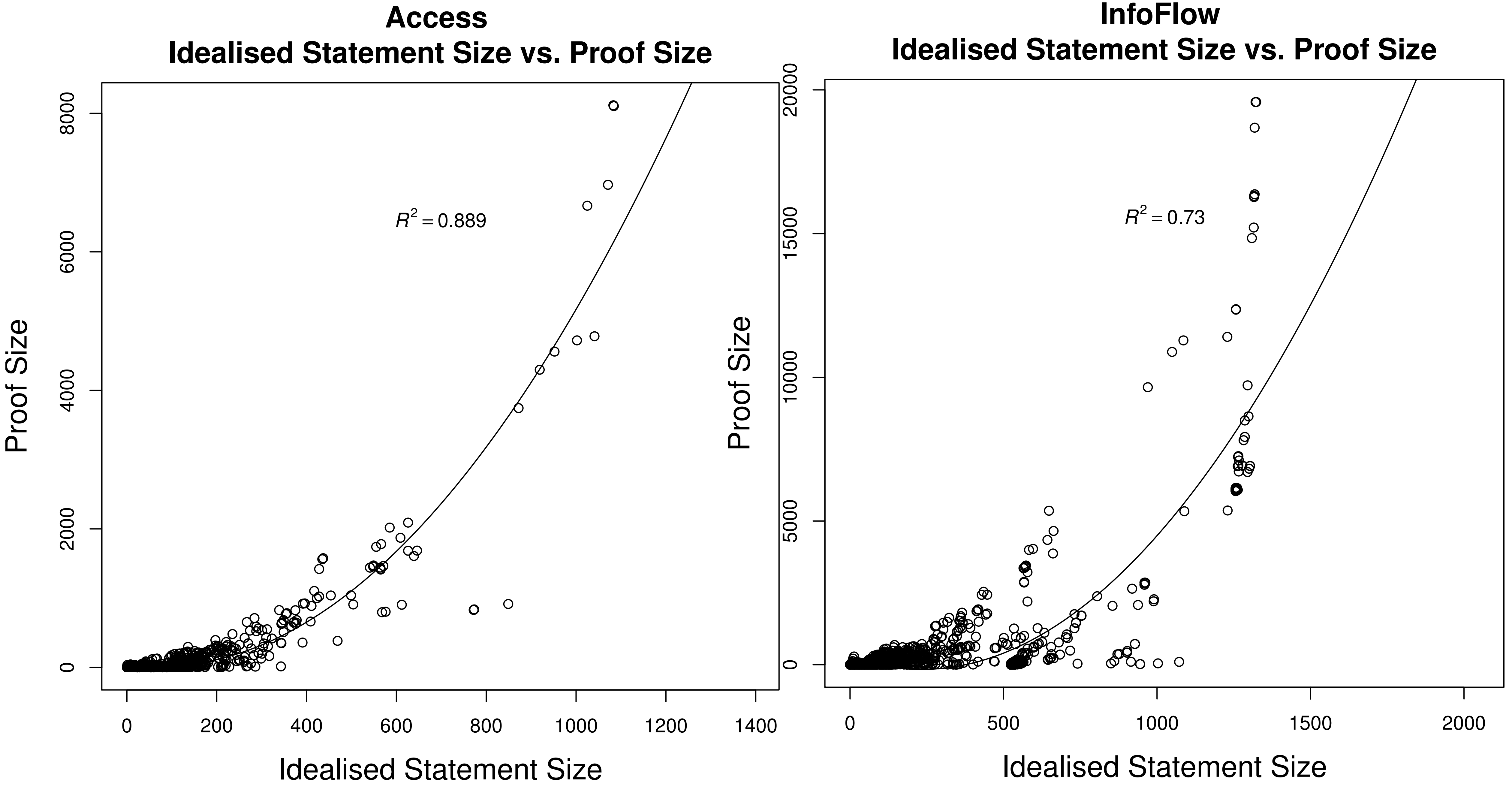


Refine
Idealised Statement Size vs. Proof Size

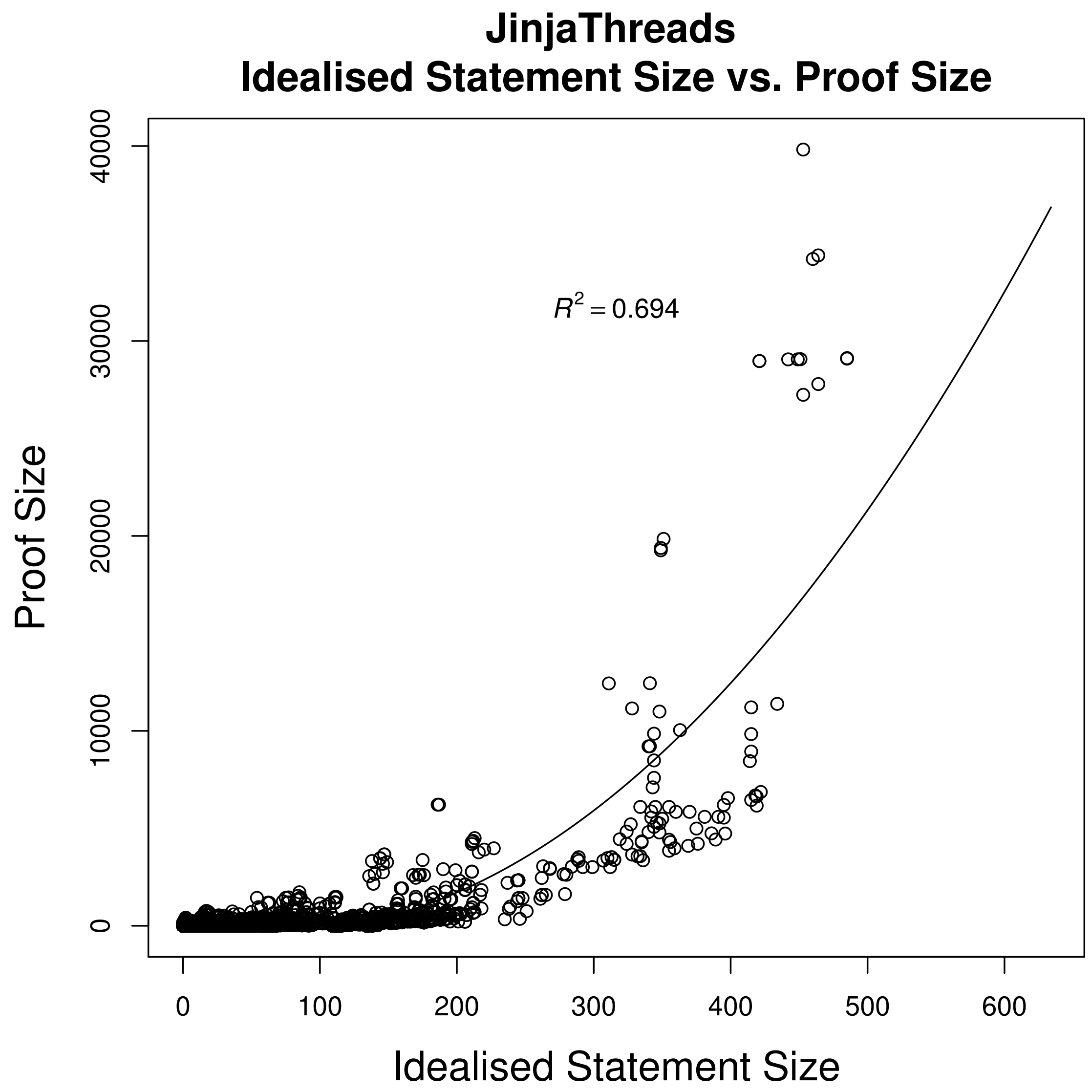
Strong Quadratic Correlation

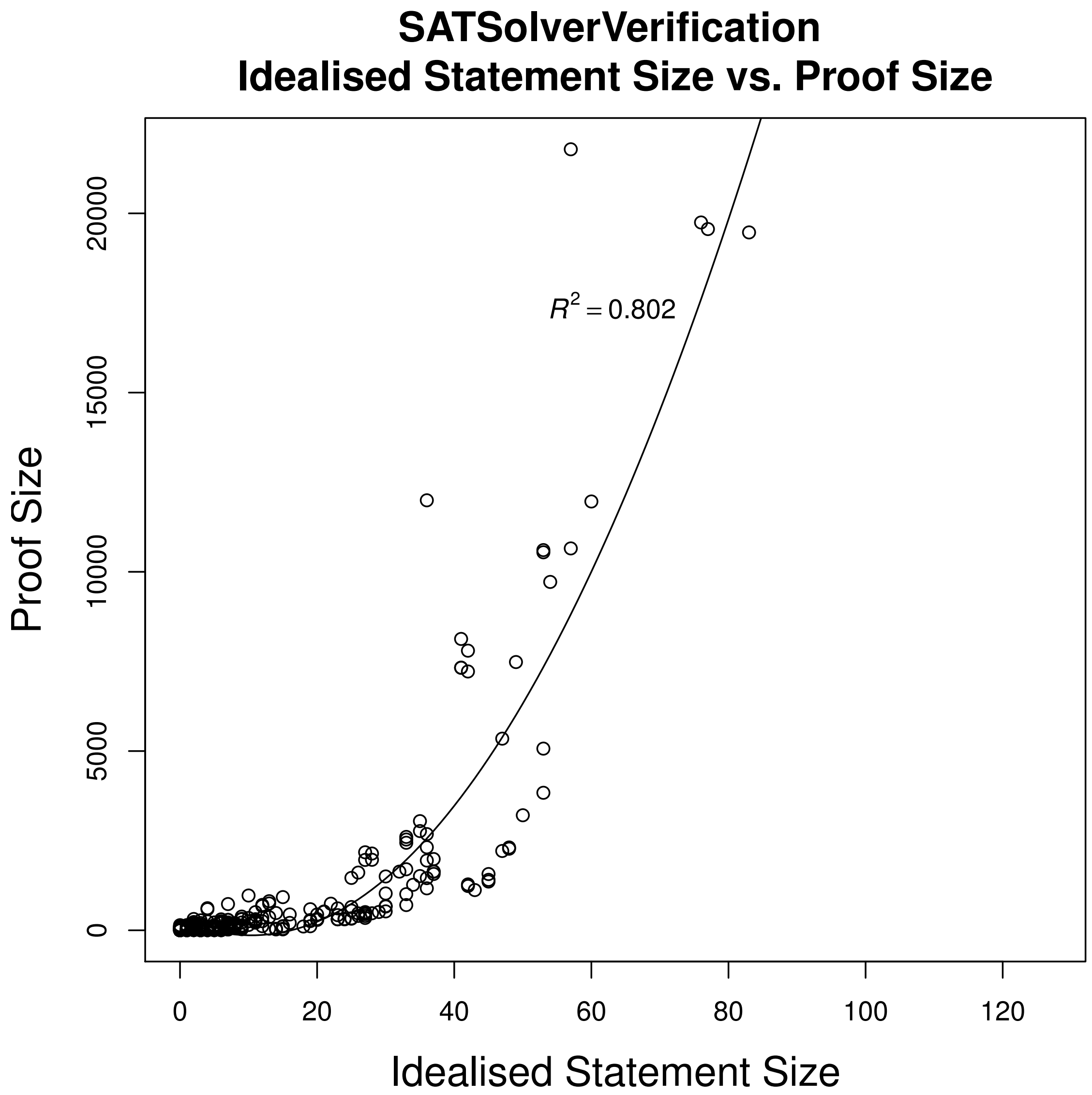
Idealised **constants**

Spec Size vs Proof Size



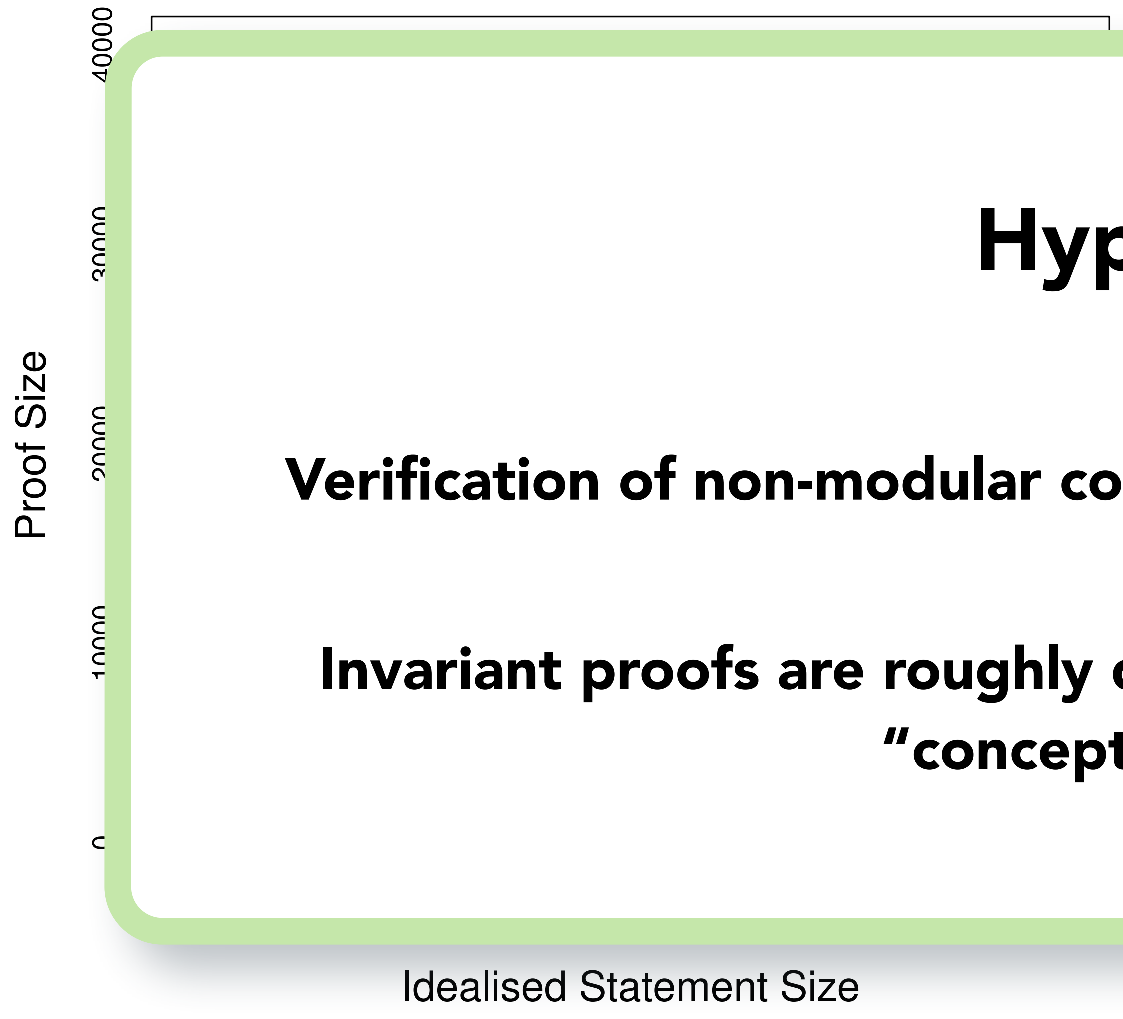
AFP Proofs



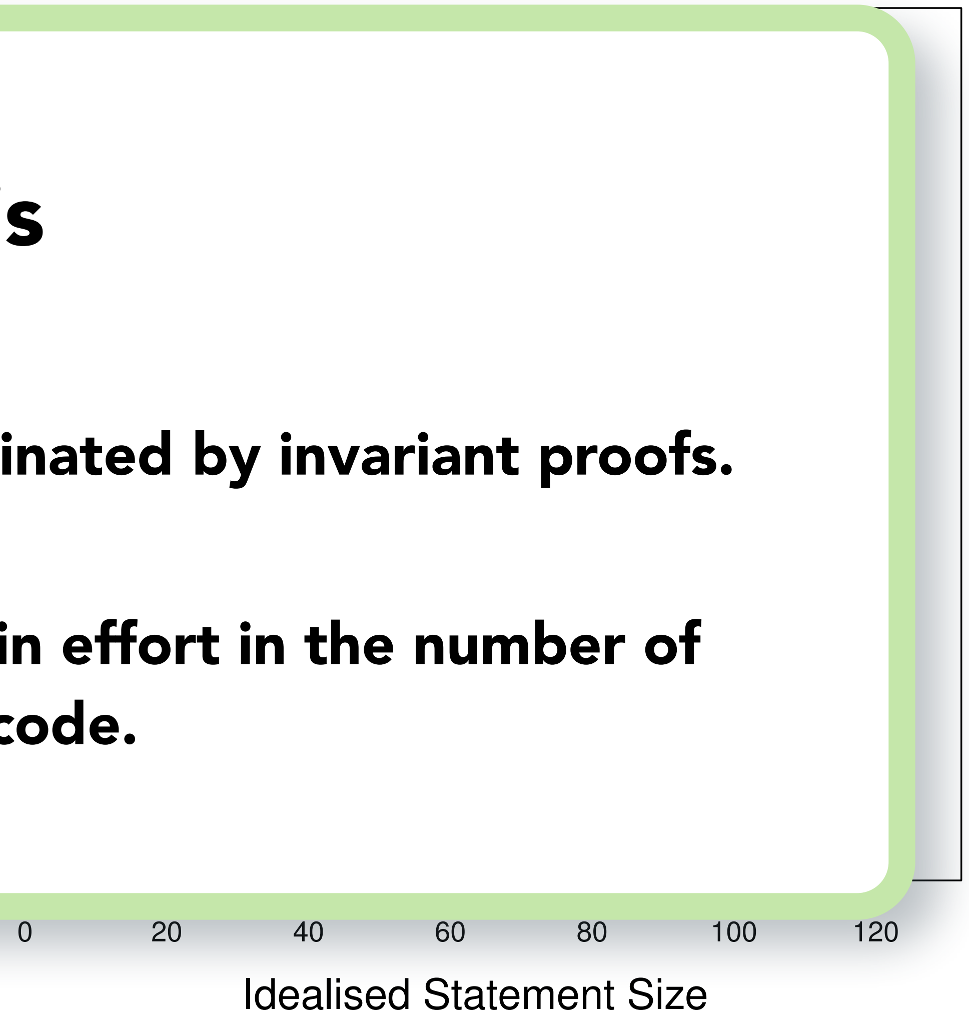


Also works for some large AFP proofs. But not all.

AFP Proofs



JinjaThreads
Idealised Statement Size vs. Proof Size



SATSolverVerification
Idealised Statement Size vs. Proof Size

Hypothesis

Verification of non-modular code is dominated by invariant proofs.

Invariant proofs are roughly quadratic in effort in the number of "concepts" in the code.

Also works for some large AFP proofs. But not all.

Some Hope

Code Size is correlated with Spec Size

Some Hope

Code Size is correlated with Spec Size

Spec Size is correlated with Proof Size

Some Hope

Code Size is correlated with Spec Size

Spec Size is correlated with Proof Size

Proof Size is correlated with Effort

Some Hope

Code Size is correlated with Spec Size

Spec Siz

Proof

There may be hope for a prediction model.

Probably applies to verification of non-modular code.

Unlikely to work for other kinds of proofs, but likely to transfer to other interactive provers.

Summary

seL4

- Full verification. Full performance.
- Already cost effective for high assurance.
- Open source and open proof.

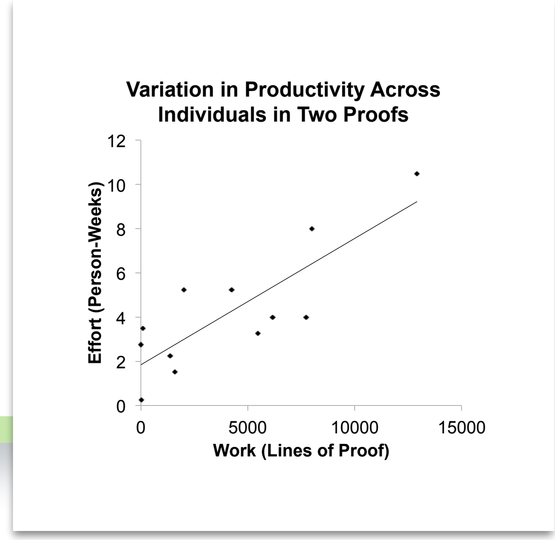
Summary

seL4

- Full verification. Full performance.
- Already cost effective for high assurance.
- Open source and open proof.

Proof Engineering

- Should become a research discipline.
- Work has started. A lot more to be done.



Thank You

NICTA Software Systems Research Group

NICTA

Thank You

SSRG@NICTA

NICTA Software Systems Research Group